UNIVERSITÁ DI PISA
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

NOVEL METHODS AND ALGORITHMS FOR PRESENTING 3D SCENES

DOCTORAL THESIS

Author
Andrea Baldacci

Tutor (s)
Prof. Marco Avvenuti

Dr. Fabio Ganovelli

The Coordinator of the PhD Program
Prof. Andrea Nannini

Pisa, November 2016

Cycle XXVIII

To my parents, for their love and support

"A man who is a master of patience is master of everything else"
George Savile

Acknowledgements

In first instance, I would like to deeply thank Dr. Fabio Ganovelli, my supervisor at the Visual Computing Lab (ISTI-CNR), for his guide though this research.

A big thanks goes also to Dr. Massimiliano Corsini for his help during these years.

Finally, my sincere gratitude goes to Prof. Marco Avvenuti, my supervisor at the University of Pisa, for its availability and support.

Ringraziamenti

Innanzitutto, vorrei ringraziare il Dr. Fabio Ganovelli, mio tutor al Visual Computing Lab (ISTI-CNR), per la sua guida durante questi anni di ricerca.

Un grande ringraziamento va anche al Dr. Massimiliano Corsini per il suo aiuto in questi anni.

Infine, la mia sincera gratitudine va al Prof. Marco Avvenuti, mio tutor all'Università di Pisa, per il suo supporto e disponibilità.

Summary

In recent years, improvements in the acquisition and creation of 3D models gave rise to an increasing availability of 3D content and to a widening of the audience such content is created for, which brought into focus the need for effective ways to visualize and interact with it.

Until recently, the task of virtual inspection of a 3D object or navigation inside a 3D scene was carried out by using human machine interaction (HMI) metaphors controlled through mouse and keyboard events.

However, this interaction approach may be cumbersome for the general audience. Furthermore, the inception and spread of touch-based mobile devices, such as smartphones and tablets, redefined the interaction problem entirely, since neither mouse nor keyboards are available anymore. The problem is made even worse by the fact that these devices are typically lower power if compared to desktop machines, while high-quality rendering is a computationally intensive task.

In this thesis, we present a series of novel methods for the easy presentation of 3D content both when it is already available in a digitized form and when it must be acquired from the real world by image-based techniques. In the first case, we propose a method which takes as input the 3D scene of interest and an example video, and it automatically produces a video of the input scene that resembles the given video example. In other words, our algorithm allows the user to replicate an existing video, for example, a video created by a professional animator, on a different 3D scene.

In the context of image-based techniques, exploiting the inherent spatial organization of photographs taken for the 3D reconstruction of a scene, we propose an intuitive interface for the smooth stereoscopic navigation of the acquired scene providing an immersive experience without the need of a complete 3D reconstruction.

Finally, we propose an interactive framework for improving low-quality 3D reconstructions obtained through image-based reconstruction algorithms. Using few strokes on the input images, the user can specify high-level geometric hints to improve incomplete or noisy reconstructions which are caused by various quite common conditions often arising for objects such as buildings, streets and numerous other human-made functional elements.

Sommario

Negli ultimi anni, il miglioramento delle tecniche per l'acquisizione e creazione di modelli 3D ha dato luogo a una maggiore disponibilità di contenuti e a un ampliamento del pubblico a cui sono destinati. Ció ha messo in luce la necessità di sviluppare metodi semplici ed efficaci di visualizzazione di tali contenuti e d'interazione con essi. Fino a poco tempo fa, l'ispezione di un oggetto 3D o la navigazione all'interno di una scena virtuale venivano effettuati utilizzando metafore di interazione uomo-macchina (HMI), controllate attraverso i comandi inviati da mouse e tastiera. Tuttavia, questa modalità d'interazione risulta essere poco pratica per l'utente comune. Inoltre, la nascita e la diffusione di dispositivi mobili basati sul tocco, come smartphone e tablet, ha ridefinito del tutto il problema di interazione, dal momento che né il mouse né tastiera sono in questo caso disponibili. Il problema è poi aggravato dal fatto che tali dispositivi hanno una potenza computazionale inferiore rispetto a macchine desktop, mentre ottenere un rendering di alta qualità richiede è un processo computazionalmente oneroso.

In questa tesi, proponiamo una serie di nuovi metodi per presentare in maniera intuitiva contenuti 3D, sia quando questi sono già disponibili in forma digitalizzata, sia quando debbano essere acquisiti dal mondo reale con tecniche *image-based*. A questo proposito vengono illustrati tre diversi metodi. Nel primo metodo, si propone un algoritmo che prende in input una scena 3D e un video di esempio, e produce automaticamente un video della scena di input analogo all'esempio video fornito. In altre parole, l'algoritmo consente all'utente di replicare un video esistente, per esempio creato da un animatore professionista, su una diversa scena 3D. Nel secondo metodo, nel quadro delle tecniche *image-based*, sfruttando l'organizzazione spaziale di fotografie scattate per ricostruire una scena, proponiamo un'interfaccia intuitiva per la navigazione stereoscopica della scena acquisita. Tale sistema riesce a fornire un'esperienza tridimensionale senza la necessità di ricostruire effettivamente la geometria della scena. Infine, nel terzo metodo, proponiamo un sistema interattivo per migliorare le ricostruzioni 3D di bassa qualità ottenute attraverso algoritmi di ricostruzione da immagini. Tracciando alcuni tratti sulle immagini di input, l'utente può fornire dei vincoli ad alto livello sulla natura geometrica delle superfici della scena. Ció permette di migliorare ricostruzioni incomplete o rumorose, che spesso si ottengo per molti scenari urbani o artefatti a causa delle caratteristiche di tali superfici.

List of publications

International Journals

1. Baldacci, A., Ganovelli, F., Corsini, M., & Scopigno, R. (2016). Presentation of 3D Scenes through Video Example. IEEE Transactions on Visualization and Computer Graphics.

2. Baldacci, A., Kamenicky, R., Riecicky, A., Cignoni, P., Durikovic, R., Scopigno, R., & Madaras, M. (2016). GPU-based approaches for shape diameter function computation and its applications focused on skeleton extraction. Computers& Graphics, 59, 151-159.

3. Baldacci, A., Bernabei, D., Corsini, M., Ganovelli, F., & Scopigno, R. (2015). 3D reconstruction for featureless scenes with curvature hints. The Visual Computer, 1-16.

International Conferences/Workshops with Peer Review

1. Baldacci, A., Ganovelli, F., Corsini, M., & Scopigno, R. (2014). Stereo-browsing from Calibrated Cameras, Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference, Cagliari.

Contents

CHAPTER 1

Introduction

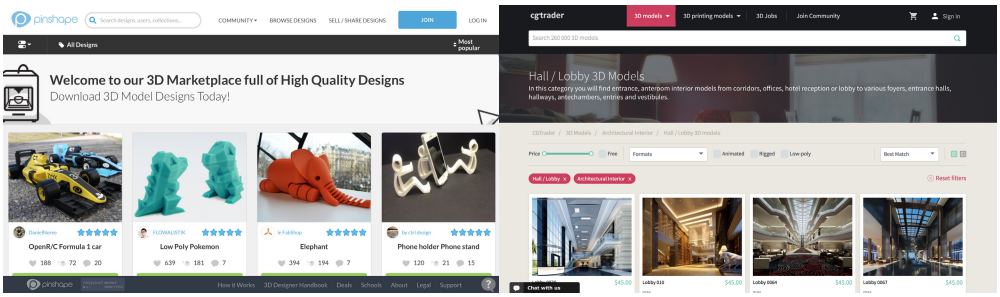
(a) (b) (c)

Figure 1.1: *Some fields of application for 3D presentations. Figure (a) shows a Ducati spot with an "explosion" animation of one of their engines. Figure (b) shows an architectural design by the Renzo Piano Building Workshop. Figure (c) shows an interactive table situated in the British Museum of London. The table shows a CT scan of the Gebelein Man, one of the six mummified corpses discovered at the end of the 19th century by Wallis Budge and dating to approximately 3400 BC.*

1.1 Motivation

The use of 3D data has become pervasive in industrial and consumer markets. This evolution is due to several factors. The advancements in 3D shapes acquisition technologies and the improvement of 3D design applications were undoubtedly two of the main causes. 3D scanners and image-based algorithms are now affordable tools to acquire 3D content from the real world. In particular, the Structure-From-Motion (SFM) pipeline gained a lot of attention as it allows the fully automatic reconstruction of 3D objects from simple photographs. 3D modeling applications improved a lot in terms of usability, passing from the first Computer-Aided-Design (CAD) tools to more user-friendly and intuitive applications like Google SketchUp. 3D graphics has thus become

a fundamental tool in many fields (see, for instance, Figure 1.1), and it is used not only as a technical tool but as a new media for communicating ideas, discoveries, projects and so on. The increased popularity of 3D graphics is also due to the rise of several communities of enthusiasts who have fun creating new contents for video games, and, more recently, 3D printing. The growing amount of data also brought to the creation of several sharing repositories, some of which are shown in Figure 1.2. Despite these developments, handling 3D contents is still challenging with respect to other forms of media like photos and videos. The taking, editing, and fruition of photographs and videos are now suitable for all with just a few clicks on a mobile phone, but, unfortunately, the same is not yet true for 3D contents which still require a fair amount of expertise to be used.
In this thesis, we tackle the problem of providing automatic or semi-automatic methods to quickly present 3D content, dealing with both data already available as a 3D model and data to be acquired from photographs.



(a) (b)

Figure 1.2: *Figure (a) shows the Web site Pinshape, here 3D printing enthusiasts share their designs to be printed. Figure (b) shows instead a screenshot of CgTrader a marketplace to buy models for games, 3D printing and architectural design.*

3D presentation of digitized data Presenting 3D contents is a frequent need for architects, designers, engineers or Cultural Heritage professionals. When dealing with already digitized data, we can think of the simplest form of presentation as just sharing the file containing the 3D data with the recipient of the presentation which must be able of visualizing it with specific software. This approach, however, narrows down the possible audience and the effectiveness of the presentation for several problems:

- Professional software can be expensive both in terms of money and computational power (they in fact usually runs on desktop platforms)
- The user interface for visualizing the 3D data is cumbersome to use
- Being a manual approach, it does not scale when one has to inspect several models
- There is no simple way of creating artistic or appealing presentation, for example, to advertise a product

When there is no need of a high-quality rendering, a viable solution to the first problem, it is the use of Web browsers which are now capable of performing 3D rendering

without requiring any additional plug-in and are seamlessly supported on low-power devices such as smartphones. A prominent example is the Web site SketchFab, which is now a popular publishing platform for 3D creations. The most problematic aspects however remain. The interactive inspection of 3D objects or the navigation inside a 3D scene with mouse and keyboard is a cumbersome and non-intuitive task for most people and the introduction of touch-based devices made this problem even worse. Moreover, the creation of professional presentations, for example, to advertise a new product, has higher requirements in artistic terms with respect to interaction modes offered by most software.
In these cases, a video produced offline is often preferable because it can make the most of the 3D model and frees the user from interacting with complex interfaces. A video presentation can be produced using the animation and rendering tools usually available in the same software package used for modeling the scene. To create a video, the designer has to set up the lighting, the materials and last, but not least, the virtual camera trajectories. This process is not only costly in terms of time and money, but it also requires a fair amount of expertise. Moreover, scalability remains an issue. If a designer had a portfolio of creations, the time to create an excellent video presentation for each item would be overwhelming.

For these reasons, various methods for the automatic presentation of 3D models have been proposed. In the first instance, much effort has been devoted to devise methods to select salient views of 3D objects. One or few images are rendered from these viewpoints and used to present the object either using a single image or a short video. Only few methods try to deal with complex 3D environments. In this case, the primary approach is trying to automatically build camera trajectories following cinematographic rules and/or other constraints.

In this thesis, we propose a method which allows the user to replicate an existing video, for example, a video created by a professional animator, on a different 3D scene. Our key insight is that two video sequences are similar if their optical flow is similar. This is especially true when shooting static environments. Therefore, we devised an algorithm that, given an input video and a scene of interest, produces a video of the input scene with an optical flow which is similar to the optical flow of the input video.

3D presentation from images In several situations, it is not necessary or even possible to have an accurate 3D model of a scene. For example, the manual 3D modeling of large scale scenes, like an entire city, is prohibitive. By contrast, the reconstruction of 3D objects from the real world using photographs has recently improved a lot and it is fully automatic. However, large scale reconstructions are still an issue. Moreover, these methods can output a 3D point cloud effectively, while the automatic reconstruction of a final 3D surface is more involved and prone to errors. Luckily, for several applications it is not quite important to have an underlying 3D data to make an effective presentation, it is instead more relevant the final user experience, which should be immersive and provide the feeling of being able to move freely (or with some reasonable constraints) inside the environment of interest. Virtual tourism is an example of

such field of applications, and Microsoft PhotoTourism, shown in Figure 1.3, one of its most popular implementation. This system exploits the large photo collections which are now common on the Web, for example on the Web site Flickr, to empower the user with a functional and immersive interface for the navigation of a location of interest, for instance, the Great Wall of China shown in Figure 1.3. The user can view how cameras were spatially organized when the images were taken, he can pick some of them, or he can take a virtual tour which consists of displaying one photograph at a time and move to the next with a transition effect which tries to give the sensation of moving inside the environment.

In this thesis, we make a step forward in this type of systems by proposing a stereoscopy-enhanced framework for the exploration of the object/location depicted in an image collection. Exploiting the spatial organization of photographs, we first search for pairs of images which are suitable to build a stereo pair. We generalize this discreet domain to a continuous one using ad-hoc interpolation strategies, and we propose an intuitive interface to navigate this space.

Finally, when reconstructing a final 3D surface is mandatory, our contribution is a novel interactive framework where the user can specify high-level geometric hints to improve reconstructions obtained with automatic image-based engines.

Figure 1.3: *A view of Microsoft PhotoTurism application showing several photographs of the Great Wall of China.*

1.2 3D content presentation

We present three novel approaches for the easy presentation of 3D content both when it is already available in a digitized form and when it must be acquired from the real world by image-based techniques. The following sections are meant to give a general background knowledge to understand our contribution which we illustrate at the end of each section.

1.2.1 3D Presentation using videos

Producing professional videos to present a 3D scene is a time-consuming task and, to obtain high-quality results, the support of a filmmaker/computer animation expert is necessary. In Chapter 2, we present a method for the automatic creation of a presentation video given in input an example video and the 3D scene to present.

Background Knowledge

The two usual standards to manually explore a 3D model are the so-called "world-in-hand" and "camera-in-hand" paradigms. In the world-in-hand paradigm, the user is supposed to inspect a small object as if it was holding it. The tool which is commonly used to implement such model is the virtual trackball. The purpose of the virtual trackball is to rotate or translate the camera view and, optionally, to apply a uniform scale factor, just using the mouse interface and a graphical widget. The trackball's most common use, however, is to orbit around an object to gain an overall understanding of its shape. The virtual trackball is commonly used in 3D modeling and processing tools, like Autodesk 3DS Max, Rhino, Blender, MeshLab and many others. The virtual trackball implemented in MeshLab is shown in Figure 1.4.

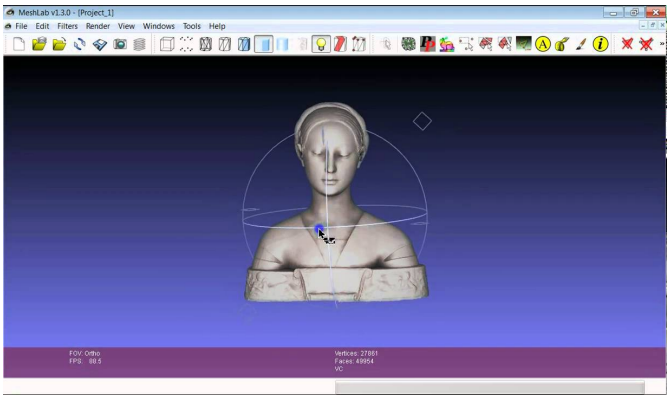


Figure 1.4: *The "world-in-hand" paradigm implemented via the use of a virtual trackball in MeshLab, one of the most used software for mesh processing.*

The "camera-in-hand" paradigm is typical in video-games, like first person shooter (FPS). In this case, the user directly impersonates the camera and can "walk" inside the scene. The camera can be translated using the keyboard and can be rotated using the mouse. A world-famous example of such approach is shown in Figure 1.5.
Manual inspection tools usually require access to the file containing the data and are clearly practical only when the number of 3D models is relatively small. The number of the available 3D models has instead grown a lot recently. Large databases are now available on-line, for example: TurboSquid, Archive3D, CGTrader. Their contents are either downloadable for free or sold. This evolution has posed two main new challenges for the research community: devising new algorithms for fast mesh retrieval from large datasets and building concise presentations visualizing salient views of the mesh to the user. This latter problem brought to definition of several principles and heuristics to craft a presentation which is maximally informative while taking the minimum amount

Figure 1.5: *The use of the "camera-in-hand" paradigm in the game Doom, one of the most famous videogame of all time.*

of time to be understood by the user [100]. In the first instance, efforts have been made to select the most "salient" view of a mesh and then presenting it with just an image rendered from that viewpoint. The saliency of a view has been defined in terms of several attributes either connected in qualitative or quantitative terms to the surface of the object under view [103]. Methods like [119] and [80], for example, give more visual importance to surface points with high curvature, either expressed as mean or Gaussian curvature. After defining the saliency of a surface region in terms of its mean curvature at multiple levels of detail, Lee et al. [80] used a gradient-descent-based optimization to seek the viewpoint that maximizes the sum of the saliency for the visible surface of the object. Another common criterion to generate the best view of an object is to select the viewpoint that maximizes the amount of surface area which is visible from that point [112]. An important user study by Secord et al. [103] shows that users select the best view basing on different criteria and it is difficult to choose one that fits for all objects. Because of this, they propose a model based on a weighted combination of attributes (projected area, silhouette length, etc.) where each weight factor was derived by fitting their model to the results of the user study.

As a single viewpoint may be not sufficient to give a satisfactory understanding of an object, some papers address the issue of the "best fly" problem. Here the issue is to select the minimum amount of necessary viewpoints and to find the optimal trajectory around the mesh which visits each view, preferably just once, and ensuring the smoothness of the path [100]. Relevant examples of such algorithms can be found in [112], [111] and [100].

Best fly methods implicitly assume a world-in-hand paradigm, as the camera always stays outside of the object and essentially orbits around it. A more involved problem is the automatic generation of exploration paths for complex 3D environments. Here, the central insight is to take into account, together with the common saliency properties, cinematographic rules and constraints. Cinematographic techniques are a natural

choice as they are a mature and well-established communication tool among mass media technologies and they are very effective to immerse the audience into a fictional world. As more deeply connected to our work, we postpone an in-depth description of these methods in Chapter 2.

Contribution

In this work, instead of creating a video from scratch, we start from an already available real or synthetic video and, given the 3D scene to present, we automatically produce a new video of the 3D scene which is "similar" to the one provided as input. Broadly speaking, our approach makes it possible to create a video for our 3D scene like our preferred film director would. The notion of similarity is central to our approach, and since input and output video contents are generally unrelated, such a notion must be as much as possible content independent. This is because, for example, we may have the input video sequence showing a dolly-in on a green tree while our 3D scene does not contain neither trees nor green elements. Moreover, we assume to have no prior information about the input video and the scene content. Therefore, we chose to focus on the optical flow as the visual attribute to be reproduced in the output video. Optical flow combines the camera movement and the 3D scene geometry and does not imply knowledge of the scene content. Hence, we state that two video sequences are similar if their optical flow is similar. A user study supports this choice and shows that videos with a very similar optical flow are also perceived as such despite their different content. Our algorithm works as follows: we estimate the camera path of the input sequence and we compute a per-frame descriptor of the video based on its optical flow. We match these descriptors against a non-redundant database of the same optical flow descriptors extracted by pre-processing the input 3D scene. The results of this matching form the basis to relocate the estimated camera path inside the 3D scene. A complete description of the algorithm and the details of the different processing steps will be given in Chapter 2. Beside the main result of this work, the automatic generation of high-quality video sequences to present a 3D scene, this contribution also includes:

- A technique to create a database of non-redundant optical flows that can be generated for a given 3D scene.
- A new per-frame compact descriptor that can be used to search efficiently video sequences with similar optical flows.

1.2.2 3D Presentation from Images: Stereoscopic Navigation

Presenting 3D locations using image-based techniques is a viable solution for several applications like, for example, virtual tourism. In Chapter 3, we propose a novel stereoscopy-enhanced navigation system for image collections.

Background Knowledge

In the last few years, due to the explosion of multi-view data, several systems for the navigation of photo collections have been proposed. Most people is now comfortable using Google Street View [4], for example, which works mainly on panoramic images. Microsoft PhotoSynth, derived from PhotoTourism [109], and PhotoCloud [17]

are other two examples, where PhotoCloud is somewhat similar to PhotoSynth but it can visualize both high-quality triangle meshes and points clouds with the photographs aligned on this geometric data. The underlying data on which these systems build their functioning is composed of the relative position and orientation of the cameras which took the pictures and a sparse 3D point cloud recovered from the image collection provided. Such data can be automatically calculated using a Structure-From-Motion (SFM) pipeline up to an unknown scale factor. In Figure 1.6, we show the PhotoSynth interface together with the underlying point cloud.
The navigation system builds on this data to create some sort of transitions between the different images using image-based rendering (IBR) techniques. There are essentially two paradigms to deal with this problem. A first approach is trying to create as seamless as possible transition effects between the photographs like, for example, in [48]. More recently, instead, free viewpoint solutions like [22] have been proposed. In this case, the user can not only view the original images but also "in-between" views that would be seen by viewpoints not present in the actual dataset. Such novel views can be generated in several ways and in Chapter 3 we will review them.

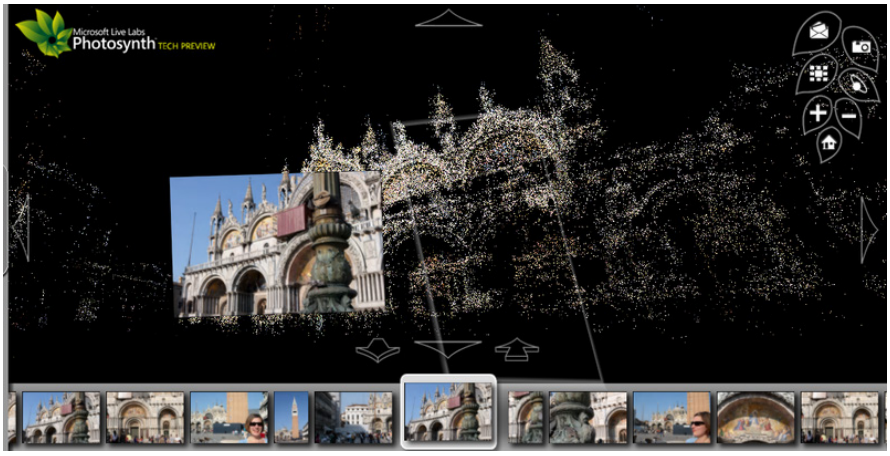


Figure 1.6: *A view of the Microsoft PhotoSynth interface. Both the images and the recovered point cloud are visible.*

Contribution

In this work, we propose a visualization system which allows the stereoscopic navigation of user photo collections. The proposed system requires a set of calibrated images as from the output of any SfM algorithm. Usual navigation interfaces neglect to use the inherent nature of such set of cameras which, despite having a wide-baseline between neighboring cameras, are often well organized. For example, photographs taken for 3D reconstruction tend to follow circular or semi-circular arcs around the subject and they are usually taken from an approximately constant height smoothly varying the orientation of the camera to guarantee a good overlap between photographs. It is thus quite likely that, among a given collection, we can find a certain number of pairs of cameras that could be rectified and used as a stereo pair. Our approach consists in extending

this discrete domain of stereo views to a continuous one by employing known IBR techniques to generate new views. We will call this domain *StereoSpace* and we will define how it can be intuitively browsed using a simple pan/zoom interface to provide a seamless stereoscopic navigation experience. Here our contribution is twofold:

- A formal definition of the space of the stereo pairs, called *StereoSpace*, which can be generated interpolating the actual viewpoints given a set of calibrated cameras.
- The prototype of a visualization system for the stereoscopic exploration of photo collections using stereoscopic devices of any type (e.g. shutter glasses, anaglyphs, etc.) that provides an immersive experience without the need of a complete 3D reconstruction of the location/object of interest.

1.2.3 3D Presentation from Images: Geometry Reconstruction

Automatic reconstruction engines have made a lot of step forwards. However, in several cases, these systems would benefit from a minimal user intervention to provide some hints about the surface to be reconstructed. In Chapter 4, we propose our user-assisted pipeline to improve incomplete or noisy surfaces obtained using automatic methods.

Background Knowledge

Image-based 3D reconstruction includes all techniques that employ images to infer the 3D shape of an object, e.g. shape-from-silhouette, shape-from-shading and multi-view stereo 3D reconstruction. Recently, multi-view stereo techniques (MVS) have improved a lot with respect to others and have become a standard tool for 3D object acquisition enabling a complex scene to be rapidly reconstructed from a set of digital images on a consumer PC. Starting from a set of input images, MVS methods build their functioning on top of the registered cameras obtained in output from an SfM pipeline. Starting from this data, MVS algorithms allow to densify the SFM sparse point cloud and to finally extract a 3D surface. An example of such software is VisualSFM [121, 123], which first calibrates the cameras and then it produces a dense reconstruction using the PMVS algorithm [42]. An analogous online service is the Autodesk 123D Catch Web application. One weakness of most of the MVS algorithms is that the quality of the final reconstruction depends on some assumptions that are not always met. Incomplete or noisy reconstruction can be caused by various quite common conditions, such as a few overlaps between images, camera movements that provide insufficient parallax information, homogeneous color appearance (lack of texture) of the object to be reconstructed, hard shadows, and moving occluders such as cars or people. These unfavorable conditions often arise for human-made *functional* elements, such as buildings, streets, bridges, pipes, toys, etc., which often consist of smooth/flat surfaces of a homogeneous appearance. To account for these and similar problems, many algorithms use different *shape priors* to increase the quality of the final reconstruction. For example, Sinha et al. and Gallup et al. [46, 108] assume that the surfaces in the scene are flat, while Furukawa et al. [41] assume that adjacent surfaces are flat and form a quasi-right angle (the so-called *Manhattan world* assumption). More complex shape priors may be used, such as the swept surfaces adopted by Changchang Wu et al. [122] for the reconstruction of architectural buildings. Another method to improve the reconstruction of missing/incomplete parts specific for architectural buildings is the

one of Chen et al. [131] which approximate the building surface with planar and curve surfaces in a robust fashion, taking into account that, in general, in this case many of the reconstructed points are inliers. However, there are cases where many different hypotheses for the missing surface may be consistent with the data. In these cases, additional geometric constraints are not sufficient and a *semantic* interpretation of the images is necessary. Bao et al. [11] introduce *semantic priors*, that is, high-level priors (e.g. a car) which are extracted and collected in a learning phase.

Contribution

We propose an interactive framework to enable the user to provide high-level geometric information to improve the reconstruction. The user can specify with a few strokes the object of interest and the geometric constraints on one or more images, i.e. parts of the object that have one or more directions of zero curvature. These hints are not helpful where the reconstructed surface is something like a tree or a bas-relief, but they are very useful for human-made objects such as walls, pipes, panels, toys, etc. For example, if a region is flat, the user can draw two orthogonal straight lines, if the region follows a cylindrical path, as in a pipe, the user can draw a single line along the pipe, and so on. The object selection is accurately propagated between all the images and on the reconstructed 3D points by a novel multi-view segmentation algorithm using a joint 2D-3D graph-cut formulation. This algorithm can be seen as an extension of the GrabCut [98] algorithm. The final surface is expressed as a union of depth maps, one for each calibrated image, and is obtained by recasting the segmentation and the curvature hints into an energy-based multi-view reconstruction problem, which is solved entirely in GPU. Each depth map is computed by minimizing an energy functional which takes into account the user indications, the initial reconstruction and the coherence among different overlapping depth maps. The multi-view segmentation and the soft constrained energy-based multi-view reconstruction algorithm of the framework are the main novel contributions of this work.

CHAPTER 2

3D Presentation using videos

2.1 Overview

In this chapter [1], we focus on the presentation of 3D models using videos and in particular on our novel contribution in this field as introduced in Section 1.2.1. In a nutshell, our method automatically generates a video presentation of a 3D scene starting from an example video that the user wants to mimic on its scene of interest which is generally different from the one depicted in the example video. The basis of our approach is our video similarity criteria based on optical flow, which effectively combines in a unique visual attribute the camera movement and the scene geometry allowing us to abstract from both the semantic of scene content and its color appearance.

In summary, this chapter makes the following contributions:

- Given a 3D scene, we show how to generate a non-redundant database of optical flows.
- We introduce a novel strategy to index and efficiently retrieve flow fields from the above database.
- Given these tools, we present an efficient algorithm to generate a video presentation of the input scene which is similar to the input video taken as an example.
- To support the choice of optical flow as our similarity criteria, we present the results of a user-study on video similarity assessment which clearly shows that optical flow plays a central role in this task.

[1] The work described in this chapter appeared in [7]

To appreciate the results of this chapter, we provide an accompanying video at the address `http://www.andreabaldacci.it/publications/#meshtrailer`.

2.2 Related Work

Our approach borrows from several fields and a complete overview of the literature related to all such fields is beyond the scope of this work. For this reason, in the following we concentrate on the works most closely related to the original contribution of this chapter, i.e.: assisted or automatic camera paths in 3D scenes, the use of cinematography in computer graphics and the metrics for estimating video similarity.

2.2.1 Assisted or Automatic Camera Paths in 3D Scenes

Several approaches share the idea of building a graph of the empty space of the scene and deriving camera trajectories from paths on this graph. In Salomon et al. [101] the graph is built by random sampling the empty space and connecting the paths which are collision-free. Andujar et al. [3] compute the graph explicitly by positioning the nodes on the medial axis of the empty space, which is approximated by voxelizing the scene and computing a distance field. A similar result is obtained by Di Benedetto et al. [28] where a *k-means*-like approach is used to distribute the graph nodes so that each point of the surface of the scene is seen by at least one node. In Oskam et al. [92] a regular grid is used of as large spheres as possible that do not intersect the model, so that each segment connecting a pair of overlapping spheres defines a collision free arc. Paths in the graph are then refined and smoothed. In Secord et al. [104] a criterion for assessing how good a specific view is given by combining a number of known other criteria [79, 96, 116] and fitting the outcome of a user test case. They were thus able to provide a path along which the view is perceptually "optimal". The main limitation of this study is that it treats small objects by only using a world-in-hand paradigm: the path is defined over a bounding sphere and the view direction is assumed to always point toward the sphere center.

2.2.2 Using Cinematographic Criteria

Using cinematographic criteria for presenting 2D-3D content is not a new idea. The seminal paper by He et al. [54] formalized concepts of cinematographic language to produce automatic videos, and a tool for shooting dynamic scenes without user assistance is also proposed. Kardan and Casanova [67] build on this approach and propose a system for the cinematographic shooting of a scene with many actors. Note that these systems, like other similar ones, assume semantic knowledge of the scene, that is, a scripted animation. Cinematography is also used in presenting 2D scenes, with the ubiquitous Ken Burns effect, which consists of combining zooming and panning action on a single image to *animate* still photographs. The technique is also used in famous films such as "La jetée"(1962), where a small trembling effect is also added to convey a stronger feeling of watching a real take. More recently, Zheng et al. [129] extended this technique by considering a small portion of a light field and adding a parallax effect without the need for segmenting front and backward parts of the image.

To some extent, these approaches pursue the same goal as we do, that is, to provide a compelling view of 3D content. Most try to include cinematographic principles and

language into the choice of a good point of view or planning the camera path. Our approach is radically different. We do not try to formalize the principles of film direction, instead we "clone" the optical flow of a specific video and transfer it to another scene in order to convey the same final impression/perception as the input video. In order to do this, we need a way to establish whether two videos are similar, a problem already raised in other fields, such as video retrieval.

2.2.3 Similarity Between Videos

Video similarity is a very general concept and various attempts have been made to reduce this concept to quantifiable criteria. Cherubini et al. [25] studied how several characteristics influence the human perception of the similarity of two videos: encoding parameters, overlay, audio commentary, photometric variations and semantic content are just a few discriminating factors. The problem mainly occurs in Content-based Video Retrieval, that is the problem of finding, in a large database of videos, the most related to a new video provided as input using only the visual information (i.e. audio information is not used). Typically, adding a video to a database requires processing it, segmenting it into sequences, to extract low-level features and creating a video signature from these features for fast retrieval (see [62] for a recent survey). Since in our case the visual content of the input video may be very different from the 3D scene of interest, we cannot use techniques that rely on color or texture features. Our method to compare videos is close to all those video retrieval approaches that use motion features for indexing and retrieval. Fablet et al. [36] use the causal Gibbs model to represent the spatio-temporal distribution of local motion-related measurements and use a statistical framework for the retrieval. Ma and Zhang [85] generate a motion texture for each frame, which shares some similarities with the histogram-based local descriptor we propose.

A subfield of video retrieval is Near Duplicate Video Retrieval (NDVR) (see [84] for a recent survey). NDVR is used for detecting copyright infringement and redundancy in large video databases such as YouTube, and the problem lies in finding when two videos are "essentially" the same. Motion estimation has proven to be a valuable feature also in this context. The method proposed by Hampapur et al. [53], splits the frame into sub-images, quantizes the motion direction of each sub-image, then uses the distribution of the vector on the possible directions as a per-frame signature. Instead of considering static cells, Hoad and Zobel [55] detect the motion of the centroids of the brightest and darkest areas of the frames and then use the distance between these centroids as a signature for the key frames.

2.3 Our Approach at a Glance

Our approach relies on a function $D(A, B)$, described in detail later in this section, which defines the distance between two videos of equal length. With this function we state our problem as: *given an input video A and a 3D scene S, produce a video B of S that minimizes $D(A, B)$.*

Since the video we are looking for will be produced by moving a camera along a camera path inside the scene of interest, we can re-formulate the problem in terms of camera paths:

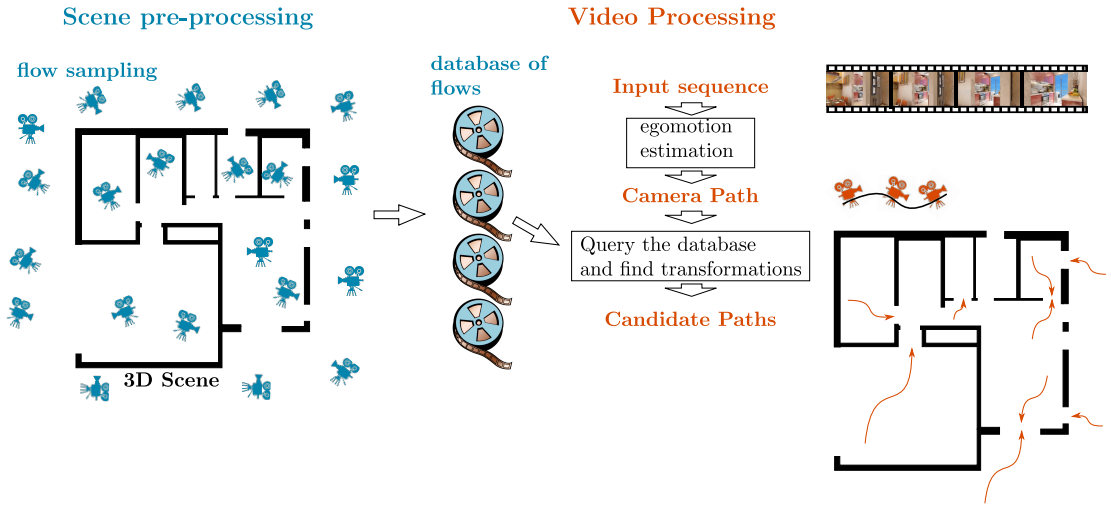


Figure 2.1: *The two stages of our approach: scene pre-processing and video processing. The pre-processing phase consists of a sparse sampling of all the possible motion fields that can be obtained by moving a camera in the input scene and efficiently storing them in a database. This phase does not depend on the input video and thus is performed once for each 3D model. Video processing starts by extracting the optical flows from the input video and its estimated camera path. With this information we query the pre-calculated database and then transform the input path to best fit the most similar scene flows.*

$$\min_{x \in \mathcal{P}} D(A, \upsilon(x)) \tag{2.1}$$

where $\mathcal{P}$ is the set of all the possible camera paths and $\upsilon(x)$ is the video produced by the path x. More specifically, we define a camera path as a discrete sequence of cameras $x = \{C_0, C_1, \ldots, C_k\}$, where a camera C_i is defined by its extrinsic parameters, that is, its position and orientation.

The brute force solution to this problem would be simply to run over all the possible paths and pick up the one with the minimum D. Clearly, the combinatorial complexity of the dense sampling of camera positions and orientations would make the problem intractable. We thus designed a pipeline which, after a pre-processing of the 3D scene, makes the problem solvable for any input video within a few minutes.

Figure 2.1 shows a high level description of our algorithm. The pre-processing phase of the scene consists of a sparse sampling of all the motion fields that can be obtained by moving a camera within the scene (see Section 2.5.1). These flows are stored in a database that can be efficiently queried to return the set of similar flows, with respect to a certain distance function ϕ, to one provided as the input (see Section 2.6 for further details). Note that this database only depends on the input scene and it needs to be created only once per model. As will be shown later, creating a database of instant flows for a typical CAD model requires a few hours.

The video processing phase takes the input video sequence and estimates the optical flows and the camera path (up to a scale factor) that produced the sequence (see Section 2.7). The domain of the problem in Equation 2.1 is then restricted to those paths obtained as a similarity transformation of the camera path estimated from the input video (see Section 2.7.1). By exploiting the database of flows built in the pre-

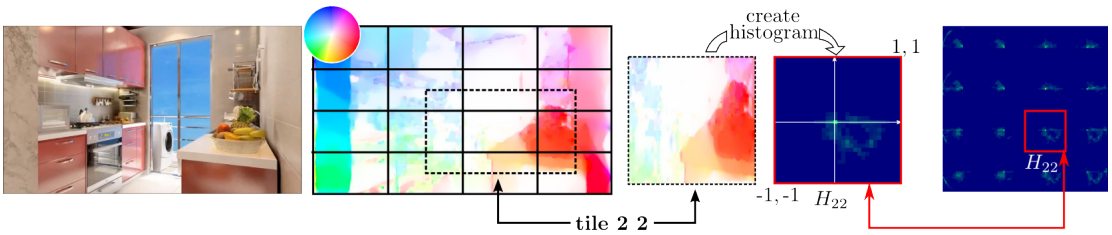


Figure 2.2: *Creation of the descriptor of vector field. The color wheel shows the color coding that maps the vectors from* $[-1,-1]\times[1,1]$ *to the RGB color. Tile* $(2,2)$ *is singled out to show how the relative distribution is stored with a histogram.*

processing phase we find a set of candidate paths. A voting scheme inspired by the Hough Transform is used to select the best candidate paths (see Section 2.7.2). Since there are many camera paths with very similar motion fields, function D may have wide plateaus. We therefore use a clustering algorithm to group the selected paths into classes of equivalence (see Section 2.7.3). Finally, we perform a non-linear optimization for refining each camera path (see Section 2.7.4). The refined path with the smallest D value is chosen as the output.

2.4 Comparing Videos by Means of Optical Flow

The notion of similarity between two videos is central to our problem. Note that we cannot assume that the input video and the video to be generated are related in terms of color and texture information. The first can be any video sequence (e.g. downloaded from the internet) and the second is a synthetic video of the scene of interest. We thus compare two videos through the movement of the camera w.r.t. to the static scene, which is captured by the motion field, that is, the 2D vector field where each vector describes the displacement of the projection of a point of the scene between two consecutive (or nearby) frames. As stated in the Introduction, the motion field of the input video is approximated by its optical flow. Since we want to produce a video of the same length as the input video, input and output videos have the same number of frames. We define the distance between two videos A and B as:

$$D(A,B)=\sum_{i=0\ldots k-1}\phi(F(A_i),F(B_i)) \tag{2.2}$$

where k is the number of frames, A_i and B_i indicates the i^{th} frame of the video A and B, respectively, $F(.)$ is a function returning the flow field and ϕ defines the distance between two vector fields as explained in Sections 2.4.1 and 2.4.2.

There are many algorithms for computing the optical flow from an input video, the Middlebury Benchmark website [5] lists about 120 different solutions. For each pixel of one image, the problem is to find the best candidate to be the projection of the same 3D point on the other image. Computing the optical flow of a video is thus prone to errors, especially in the textureless parts of the image. We evaluate the optical flow using the algorithm by Brox et al. [18] which has a good trade-off between computation time and accuracy/robustness. On the other hand, we can compute the motion field of the virtual camera in the 3D scene by re-projecting each pixel back in the 3D scene and then onto the next camera frame.

2.4.1 Flow Descriptor

To design a distance function between two 2D vector fields we must consider their nature. Since ours are produced by a moving camera in a static environment, we can expect these fields to be quite spatially coherent. For example a *camera crane* (a straight upwards panning in CG parlance) would produce a vertical vector flow pointing downwards, while a *dolly in* would produce a vector field where all vectors are radially oriented away from the image center.

In order to build our field descriptor, we first normalize it by dividing all the vectors components by their maximum over all the images. Since for real videos there may be large outliers due to mismatching pixels, these maxima are computed after a simple outlier removal step, where all the vectors larger than a factor 1.5 of the standard deviation are discarded.

Then we divide the images in regular $T \times T$ tiles. Within each tile the vector field tends to exhibit a slow varying direction and magnitude. This flow can be easily represented by a 2D histogram of the distribution of vector values H_{hk}. In other words, each histogram H_{hk} stores the distribution of vectors in the tile (h, k) in $N \times N$ bins. Our local descriptor is the set of T^2 two-dimensional histograms, one for each tile from which the image is subdivided. Figure 2.2 shows how the histogram is built considering each tile plus 50% of the neighborhood tiles. Considering overlapping tiles mitigates the aliasing effects that can be produced. For example, a small translation of the video may correspond to a very large difference in the corresponding histograms. Note that, due to removing the outliers the amount of total units per histogram may not be always the same. This is why we normalize each histogram.

2.4.2 Distance Between two 2D Vector Fields

We define the distance between two flow fields as follows:

$$\phi(a, b) = \sum_{\substack{h=0...T-1 \\ k=0...T-1}} EMD(H'(a_{hk}), H'(b_{hk})) \tag{2.3}$$

where H' is the normalized version of the histogram H and EMD is the *Earth Mover's Distance* (see Rubner et al. [99]). The EMD defines the distance between two histograms as the minimum number of units to move in order to turn one histogram into the other. In other words, it is tightly related to the amount of "work" necessary to transform the first histogram into the second one. In our implementation, equation 2.3 is efficiently computed using the L1 version of the EMD, called EMD-L1 [83], which uses the Manhattan distance instead of the standard L2 ground distance.

2.4.3 Flow Comparison and Descriptor

Our definition of the distance between flows (see equation 2.3) depends on the number of tiles in the subdivision of the image (defined by parameter T) and on the number of bins (defined by parameter N) in the histograms. These parameters affect both the value of $\phi(a, b)$ and the time to compute this value. After a few trial and error tunings, we found that values of T under 4 produce very similar descriptors for dolly-in and dolly-out camera movements, therefore failing to properly discriminate between them,

and values of N under 16 also led to noticeably different camera movements to produce similar descriptors. Finally, we found that $T = 4$ and $N = 32$ provides satisfactory results.

Since higher values of T and N require a longer processing time, we also investigated whether values higher than the selected ones could improve the descriptor. We thus created a reference set of 100 flow fields, computed by picking random consecutive frames for a video. We then chose a pivot field and sorted the other 99 in ascending order with respect to ϕ. We then repeated this sorting for different values of parameters T and N, and then compared the sequences with the original one. We used *Spearman's Rank-Order Correlation (ρ)* [65] to measure how much the ordered sequences differed. The comparison demonstrated that higher values of T and N basically produced the same results (e.g. $\rho = 0.96$ for $T = 8, N = 32$ and $\rho = 0.93$ for $T = 8, N = 64$). This confirms that $T = 4$ and $N = 32$ is a sensible choice for these parameters.

2.5 Pre-processing the Scene: a Database of Instant Flows

The basic building block of our pipeline consists of turning a 3D scene into a (large) database of motion fields. We define *instant camera* as a pair (C, Δ) where C is a camera and Δ is a small 3D displacement, and *instant flow* is the normalized motion field obtained considering the image produced by the camera C and its translated version $C + \Delta$. The length of Δ is chosen to be sufficiently small to guarantee that there is virtually no disocclusion between C and $C + \Delta$. This is achieved by rendering the scene with C and setting $\|\Delta\| = \frac{10 D_{\min}}{w\, f}$, where $D_{\min}$ is the minimum value of the depth map, w is the viewport size in pixels and f is the focal distance. This choice bounds the maximum length of a flow vector to be at most 10 pixels.

With these new definitions, the problem of creating the database of instant flows corresponds to sampling the space of our scene with a suitable set of instant cameras. Each instant camera is defined by eight variables: three for the position, three for the orientation and two for the direction of the displacement vector Δ.

A simplistic way to build the database would be to perform a regular dense sampling of the scene for the camera position, and to consider for each position a set of predefined orientations and displacements. Clearly, the number of samples would rapidly grow to an unmanageable size. For example, even considering a $4 \times 4 \times 2\ m^3$ empty room with a sampling rate of a position every 10 cm, 128 orientations and 64 displacement directions would give over 262M instant cameras. The next section describes an ad-hoc strategy to obtain an efficient time and space sampling.

2.5.1 Adaptive Sampling of the Instant Flows

Let $\mathcal{C}^p$ be the set of all instant cameras with position p, that is, that differ only in terms of their orientation and displacement, and let $S(p)$ be a panoramic depth map taken from point p. If we create the instant flow of any instant camera in $\mathcal{C}^p$ by rendering the depth map $S(p)$ instead of the actual 3D scene, the result will differ significantly only where the disocclusions are, i.e. the part of the scene visible from $p + \Delta$ and not from p (and thus not in the depth map created from p). Since Δs are small (see below), we can reasonably assume that there are few disocclusions and that their effect is negligible. It follows that if the panoramic depth maps for two points are the same,

the corresponding set of instant flows will also be the same. We can thus reduce the number of dimensions of the domain to be sampled from eight to three, that is, the camera positions. Therefore, the original problem can be re-stated as finding a sparse sampling of 3D points (with associated sets of instant flows) according to panoramic depth map differences.

The sampling process is incremental and consists of sampling the volume with a layered series of the Poisson Disk distribution with a decreasing disk radius. The samples are generated with a technique inspired by the approach by White et al. [120] for the 2D domain. We start by initializing the sampling with a radius of $R = 1/20^{\text{th}}$ of the scene bounding box diagonal. Then, at each sampling step the radius is reduced by a factor equal to 0.8. Each candidate sample at the i^{th} step is inserted if there is no sample within a radius $(0.8)^i R$ whose associated panoramic map is too similar to that associated with the candidate sample. In our implementation, the panoramic depth maps are implemented as cube maps and the similarity is taken as the mean difference of depths. Note that we add a large sphere containing all the scene to prevent the constant far-plane depth values from biasing the depth map comparison, as is done in [28].

2.6 Indexing and Retrieval

For each sampling point p, we create the set of instant cameras $\mathcal{C}^p$ and the corresponding instant flows by rendering the scene from C_i and $C_i + \Delta_i$ for all $C_i \in \mathcal{C}^p$. These instant flows are then stored in a database.

Since we need to perform many *nearest neighbours* queries on this database, we want the query to be efficient. Unfortunately, the Earth Mover's Distance, which is at the core of the definition of our measure ϕ, is not amenable to organizing the data into a hierarchical data structure. Furthermore, computing the EMD entails solving a max-flow problem, which is computationally demanding. We thus define a Hamming Embedding, i.e. a binary hashing, to map a real-valued vector containing the data of interest, to a vector of boolean values. We then use the Fast Matching algorithm by Muia and Lowe [87] for the fast retrieval of nearest neighbours with respect to the Hamming Distance. This algorithm is implemented in the FLANN library [88]. This is a classic strategy for many applications, for example in the field of image retrieval. Storing the binary descriptor instead of the original onealso enables us to reduce the size of the database by about one order of magnitude.

Obviously, it is crucial that the embedding and the Hamming distance produce similar mapping from histograms to vector of bits. Following Jéoug et al. [64] we create the embedding as follows. Each $N \times N$ histogram H_{hk} is regarded as a N^2 vector of real values. A matrix Q of m random orthogonal vectors is generated and forms a basis of a m-dimensional subspace of R^{N^2}. Then, the embedding for a tile is defined as

$$\begin{aligned} H'_Q(a_{hk}) &= Q\ H'(a_{hk}) \\ emb(a_{hk})[i] &= \begin{cases} 1 & H'_Q(a_{hk})[i] > 0 \\ 0 & H'_Q(a_{hk})[i] \leq 0 \end{cases} \end{aligned} \tag{2.4}$$

The bigger m, the more accurate the mapping, and the larger the database. In our experiments we set $m = 128$ leading to a 128 bit long binary signature for each tile. The final instant flow signature is simply obtained by concatenating the signature of

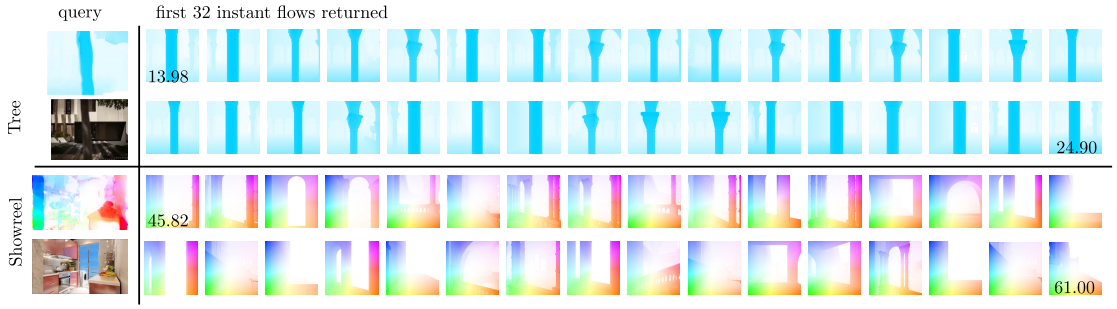


Figure 2.3: *Two queries from two separate videos on the databases: Sibenik and Museum (see Section 4.6). (Top) The input flow is produced by a camera track behind a tree. The result is a set of instant flows produced by a camera track behind various columns of the Sibenik 3D model (Bottom) The input flow is produced by entering a room. The result is a set of instant flows produced by passing through an arch of the Museum 3D model.*

each tile thus obtaining a $128 \times 4 \times 4 = 2048$ bit signature for the descriptor of each instant flow. Hence, the database size is reduced by a factor of 32, that is 2 KB instead of 64 KB (4×4 tiles, 32×32 bins).

The execution of a query is a two-step process: we first retrieve the closest 2048 neighbors according to the embedding, and then re-rank the result of this query according to the ϕ distance (that is, with the EMD distance) in order to find the most similar instant flows.

Figure 2.3 shows two examples of queries and their results on the databases: Sibenik and Museum (see Section 4.6). We also report the value of ϕ for the first and the last returned instant flows. Notice that the error range of the first query ($[13.98, 24.90]$) is much smaller than the second one ($[45.82, 61.00]$), which presents a more cluttered flow.

2.6.1 Two-step Query Accuracy

As explained above, the query of the instant flows is approximated by a two-step approach. First, we query the database of the embedding vectors. Second, we sort the results according to the function ϕ. Since there is no guarantee that the closest 2048 w.r.t. to the Hamming distance will contain the closest 128 w.r.t. to the proposed ϕ distance we evaluated the accuracy of this approximation empirically.

We compared our query result with the result obtained by a linear search in the entire database using ϕ and taking the closest 128 elements. More precisely, we measured the accuracy $\mathcal{A}$ as:

$$\mathcal{A} = 100 \, \frac{\#\{b \,|b \in \mathcal{N}(a) \;\wedge\; \phi(a,b) < \phi(a, EMD_{128})\}}{128} \tag{2.5}$$

where $\mathcal{N}(A_i)$ is the set of instant flows returned by our query and EMD_{128} is the last element of the result returned by the linear search. An accuracy of $x\%$ means that the $x\%$ of $\mathcal{A}$ is within the distance of the 128^{th} flows returned by the linear search. We evaluated the query accuracy $\mathcal{A}$ over 100 frames randomly selected among an input sequence. Eight 3D models were considered for this evaluation. For each model, a database of about 150,000 instant flow was generated. The number of instant flows was chosen so that the linear search could be done in tractable time and, at the same time is

high as in the real scenario. On the average we achieved about the 52.5% of accuracy, which is more than enough for our purposes.

2.7 Processing the Input Video

Given a shot of a static scene we can estimate, up to a scale factor, the trajectory followed by the camera that produced it. This is process is known as *camera tracking*. In out setup this is done by using the Voodoo Camera Tracker software. Given the estimated camera path $P(A)$ and the corresponding input flow $F(A)$, we can define the search space of our problem as the domain of the similarity transformations of $P(A)$. In other words, we can relocate and scale the path $P(A)$ in our scene by looking for the transformation that best replicates the flow of the input shot $F(A)$.

$$\arg\min_{\tau \in \Omega} D\left(A, v(\tau(P(A)))\right) \tag{2.6}$$

where Ω indicates the group of angle preserving affine transformations and $v(.)$ indicates the video corresponding to the given path.

2.7.1 Finding Candidate Paths

Our aim is to reduce the search space in (2.6) to those transformations that are most likely to provide a good result. We now show how this is done using the database of instant flows. Figure 2.4 illustrates an example with a path and a scene. Given a generic frame A_i, we can define the corresponding instant camera (C_{A_i}, Δ_{A_i}) where $C_{A_i} = P(A_i)$ and $\Delta_{A_i} = \nabla P(A_i)$. By querying the database for $F(A_i)$ we retrieve a set of instant cameras whose flow is similar to $F(A_i)$ that is, $\mathcal{N}(A_i)$ defined in Section 2.6.1. Each instant camera $C_h \in \mathcal{N}(A_i)$ defines a transformation RT such that $RT(P(A_i)) = C_h$, that is, the rigid transformation from the camera $P(A_i)$ to the camera C_h. This transformation is found by expressing the camera parameters as a set of 3D points and solving the point matching problem as in [59]. More precisely, each instant camera (C_h, Δ_h) is associated with one point for its origin, three along the axes, one for the direction of movement Δ_h.

Note that choosing a single point on A does not uniquely define a transformation τ, because the scale factor is undefined. In other words, Figure 2.4 highlights that there are infinite transformed versions of $P(A)$ that go through (C_h, Δ_h) in $P(A_i)$. If, instead, we choose at least two points in $P(A)$, the scale will be determined and the resulting transformation will uniquely identify a *candidate path*. In summary, for each pair of instants (A_i, A_j), we can create a set of *candidate paths* $\Gamma(i,j) = \{\tau_{hk} |\ (C_h, C_k) \in \mathcal{N}(A_i) \times \mathcal{N}(A_j)$. We recall that the sets $\mathcal{N}(A_i)$ and $\mathcal{N}(A_j)$ correspond to those instant cameras whose flow is similar to $F(A_i)$ and $F(A_j)$. Therefore, the terms of the sum in equation 2.2 will tend to be small for frames i and j. What we need is a path for which as many as possible terms of that sum are small. The problem thus becomes a fitting problem where D is the error function and where each pair of frames (A_i, A_j) produces a set of candidate paths, that is, solutions, $\Gamma(i,j)$.

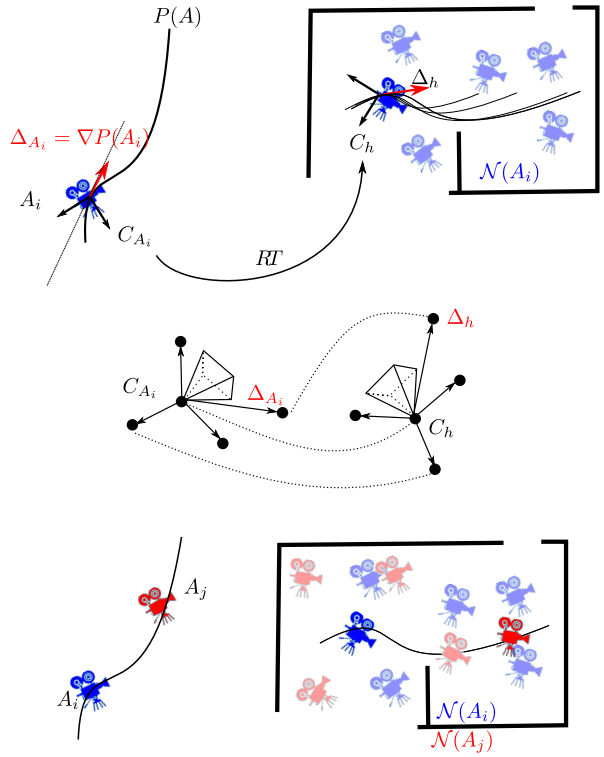


Figure 2.4: *(Top) The path $P(A)$ is estimated from the video A and an instant camera is created for frame A_i. For each instant camera $C_h \in \mathcal{N}(A_i)$ there is a rototranslation operation which brings the camera C_{A_i} on C_h, leaving the scale value undefined. (Middle) How a set of 5 points is built from an instant camera in order to find the transformation RT. (Bottom) Setting a correspondence for two instant cameras defines the scale factor.*

2.7.2 Best Paths Selection

In order to select the best paths we follow a Hough Transform-like approach [10]. Any pair of instant flows chosen along the path $P(A)$ produce a set $\Gamma(i,j)$ of candidate paths. It is likely that distinct pairs of instant flows may produce non-disjoint sets of candidate paths. Using a voting scheme we can select those candidate paths that appear in most sets. Votes are accumulated in a 7-dimensional matrix $\mathcal{M}$ where each cell corresponds to a value for the rototranslation and scaling parameters that define a candidate path. The voting scheme uses a subset of all possible pairs as follows. First the video is adaptively subsampled with respect to the change of optical flow: the higher the change the denser the sampling. We conservatively set the subsampling scheme to keep the 70% of the frames. Then we consider all the pairs (A_i, A_{i+1}), (A_i, A_{i+2}) and (A_i, A_{i+3}). This allows us to use only $3(0.7K-1)$ frames out of the $K(K-1)$ possible pair of frames without losing the local coherence of the path.

For each pair (A_i, A_j)

1. We retrieve the two sets $\mathcal{N}(A_i)$ and $\mathcal{N}(A_j)$ from the database
2. We compute the set of candidate paths $\Gamma(i,j)$
3. We quantize the elements in $\Gamma(i,j)$ thereby obtaining $\Gamma'(i,j) \subseteq \Gamma(i,j)$

4. for each element in $\Gamma'(i,j)$ we add 1 to the cell $\mathcal{M}[\Gamma'(i,j)]$ and to the 7-dimensional ball of radius 1 (in cells) centered in $\Gamma'(i,j)$

Thanks to the quantization in step 3, candidate paths that fall in the same cell in the parameters space do not lead to multiple votes. The extension of the vote to the adjacent cell in step 4 is simply to mitigate banding effects.

2.7.3 Clustering

At the end of the voting the peaks of the accumulation matrix will correspond to those paths that were voted by the greatest number of pairs of frames. As expected, there is not just one solution to our problem. The optical flow produced by a camera moving along a corridor may be replicated on each and every corridor of our 3D model. Thus usually there are many reason the peaks of matrix $\mathcal{M}$. Moreover, a set of peaks close to each other may essentially describe the same path. We thus apply a clustering algorithm to the peaks, that computes a few well defined and distinct set of output paths. In the current implementation clustering is carried out by a simple k-means on the position of the camera. We found that this is sufficient for our needs. Parameter K controls the number of distinct paths produced by the system. In our experiments we kept this parameter quite high ($K = 100$) in order to examine how the different paths explored the scene of interest. The number of distinct paths can also be estimated automatically by applying one of the methods for estimating the number of clusters of a dataset (see [125] for an overview).

2.7.4 Final Refinement

In order to create the database of instant flows, we quantized the set of all the possible orientations and displacements. To find the candidate paths, we did the same for the $7D$ space of transformations. Finally, we applied clustering to the final outcome in order to return a usable number of solutions. Although these approximations were necessary and are fine tuned, it is clear that they may lead to sub-optimal results, that is, paths which could be improved with small variations. We thus applied a final optimization procedure to each path returned by the clustering step (that is, each transformation τ in equation 2.6). We initialized equation 2.6 with the returned transformation, and ran the optimization with NEWUOA [127] (a derivative-free optimization algorithm for unconstrained minimization problems), on the 7 parameters of the similarity transformation.

Since we consider K clusters, we should solve the optimization problem K times. Instead, we first order the K path according to the distance D. Then, we only refined the first $\tilde{K} < K$ paths (we used $K = 100$ and $\tilde{K} = 5$ in our implementation). We refine only a subset to save computation time due to the fact that the refined path is quite close to the non-refined one and the order does not change significantly. The final output video was chosen as the path with the minimum distance D after the refinement. In the accompanying video we show an example of some of the generated solutions ranked according to the distance D.

2.8 Results

This section is organized in two parts. The first part reinforces the hypothesis we built our pipeline on, that is: the optical flow plays a fundamental role in perceiving video sequences as similar in static scenes. In order to do this, we show the results of a user study we conducted.

The second part shows that our algorithm is efficient in producing video sequences that are similar to the input one in terms of time and resource use. Some output videos produced with our system can be found in the accompanying material.

2.8.1 User Study

The idea behind this user study is to investigate the role of the optical flow in evaluating the similarity of two video sequences.

The test consists in showing a short reference video to the user plus other five video sequences and asking him/her to sort these five sequences from the most to the least similar to the reference video. It was not specified what *similar* means, that is, no suggestions were given concerning the criteria the user was supposed to use in sorting the sequences. We created five such tests with different models and camera movements (described below) and presented them to each subject through a web page. Figure 2.5 shows a snapshot of the web page for one test. The GUI was designed to be as straightforward as possible: when all the videos had been watched, the user could drag and drop the corresponding thumbnails to the right slot, rearranging and re-watching the videos without restrictions. All the users were presented with the same batch of five tests in a single scroll-down web page. The order in which the tests and the five videos of each test were laid out was always randomized to avoid possible bias. The total duration of the experiment was about 15-20 minutes. The user study was conducted with 47 subjects: 75% of whom were male and 25% female. Around 75% of the subjects had no specific skills in computer graphics or video-related topics. On the opening page, a short video tutorial showed users how to perform the test and a brief textual description of the study itself is provided. Personal data were collected using a form at the time of the results submission.

All the video sequences were obtained through rendering in order to have full control on their visual properties and on the camera movements.

We compared the ranking provided by the participants to that produced by our distance function ϕ. A high correlation between the two rankings indicated that the optical flow played a role in assessing video similarity.

Tests Description

The following section describes how the five tests were designed.

Test 1 - `Buildings1` In this test the reference video is an indoor sequence shot in a building, and all the other sequences are generated with similar camera paths inside a different building.

Test 2 - `Buildings2` This test is similar to the previous one, but different camera paths are used.

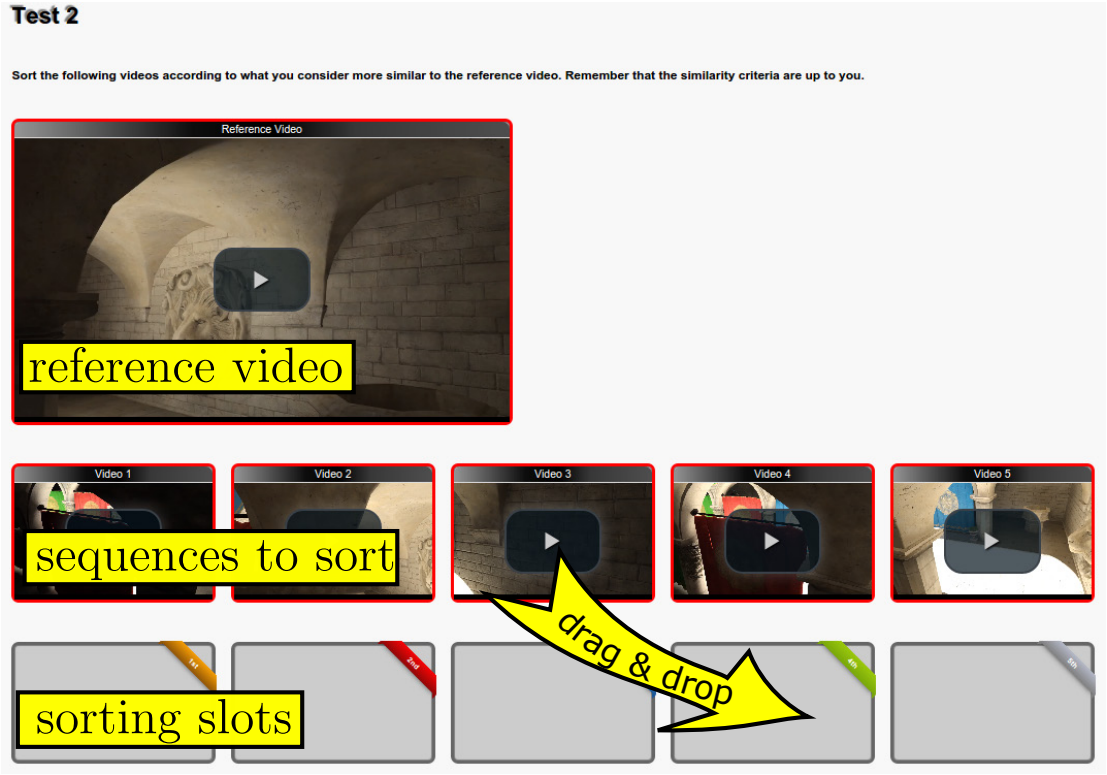


Figure 2.5: *Using simple drag-and-drop operations, the subject should rank the video sequences by similarity with respect to the reference one. The position of each sequence above the sorting slots is always randomized to avoid possible bias.*

Test 3 - `Tree` This test consists of a video shot from a camera orbiting around a tree while the others video sequences were shot with similar camera movements but looking at different subjects. This test is designed to understand whether a user to considers, videos with similar optical flows but different content as similar.

Test 4 - `Lighting` In this test all the videos refer to the same 3D model while camera paths and lighting conditions vary. The 3D model used has a large opening on the top, so we manipulated the lighting conditions by simply using the sunlight as light source and setting a different daytime for each video. Of the five videos, one has the same daytime as the reference one, other two are generated with slightly different daytimes from the reference one, which causes a small visual difference in the projected shadows. Finally, the last two have a greater difference in the daytime settings so that not only the shadows but also the overall brightness of the scene are very different. We modify the path of these videos so that the ranking of the videos according to the lighting differences is the opposite of the ranking according to the optical flow measure ϕ.

Test 5 - `ColorGrading` In this test the videos are characterized by different camera paths but also with different color grading operation (a simple hue-shifting plus contrast changes are applied). The color of the video is modified so that the ranking according to the color difference is the opposite of the ranking according to the flow field distance (more details on this point are provided later).

Test Name	Kendall τ-c	Spearman ρ
Buildings1	0.4177 (0.0000)	0.9 (0.0833)
Buildings2	0.5422 (0.0000)	0.9 (0.0833)
Tree	0.1289 (0.0187)	0.3 (0.6833)
Lighting	0.4045 (0.0000)	0.8 (0.1333)
ColorGrading	0.4756 (0.0000)	1.0 (0.0167)

Table 2.1: *Data analysis of the results. The numbers in parenthesis are the values of the corresponding Null Hypothesis (rounded to the fourth decimal place). Spearman's correlation coefficient is computed on the corresponding Normalized Rank (provided by the users).*

In summary, the first two tests consider scenes with a similar semantic content but with different camera movements. The Tree test considers a scene with different camera movements and different semantic content, and tests 4 and 5 are used to understand the impact of optical flow w.r.t other visual attributes, i.e different lighting conditions and different color gradings.

Data Analysis

Our aim was to discover whether users sort the video sequences in the same way as our flow field-based distance function ϕ does. If this is the case, it means that our distance function is successful at capturing the perceived similarity between video sequences. To put this concept into numbers we calculated, for each test, a value typically used in non-parametric statistics for measuring the correlation between two rankings: the Kendall τ-c value. This value expresses the degree of correlation between a group of orderings (provided by the users) and a reference one (provided by our distance) and ranges between -1 and 1. A value of τ-c greater than 0.4 means that the reference order has a good correlation with the orderings provided by the users, while a value greater than 0.6 means that the reference order is strongly correlated with them. A value of 0.2 identifies a weak correlation.

Table 2.1 reports the Kendall τ-c values obtained. The value in parenthesis is the probability of the null hypothesis, that is, the probability that τ-c value is not significant. For further confirmation of the results obtained, we also calculated, for each test, the Spearman rank correlation coefficient between the *normalized rank* given by the subjects and the ranking provided by the distance ϕ. The normalized rank is the overall rank which takes into account all the user data for the specific test and is obtained by following the normalization procedure indicated by Guilford [65].

Optical flow and semantic

Our approach is built on the intuition that the optical flow of a video is related both to the camera motion and to the scene content. We would thus expect two video shots with a similar optical flow to be in some way perceived as visually similar despite their 3D content. Clearly this is particular true when we treat videos and scenes with similar content while other factors may come into play when the content is very different.

Our results revealed that our approach works very well for all the tests except for the **Tree** test. Note that for the **Buildings1** and the **Buildings2** tests, where the correlation coefficients are very high, the 3D content used is similar but not that similar. In fact the layout, the geometry and the furniture of the building in the reference video is

different from the building in the comparison videos. In the **Tree** test, the correlation between subjective rankings and the optical flow distance ϕ is low as indicated by Kendall's τ (0.1289). Also the Spearman's ρ is low (and not significant, note the high p-value). This indicates that the optical flow alone is insufficient to provide perceptually similar results in this case, where very different scene content is used.

In conclusion, we can state that the results obtained support our approach although the lack of semantic information may prevent the output video from being perceived as similar to the input one in some cases. However, this does not automatically imply that the user is not happy with the result obtained, in fact our user study did not investigate the satisfaction of the user but only the perceived similarity. Adding semantic information to the 3D model (for example by annotation) and to the input video (for example by using object recognition algorithms) could be useful to enable our pipeline to produce an output video with *the same* semantic content. For example, if the input video contains a shop and we need to present the 3D model of a whole town, the candidate paths that produce videos with a shop may be preferred over others.

Optical Flow vs Other Visual Attributes

Although only two tests of this type were carried out, interesting aspect needs highlighting: the optical flow is a stronger stimulus than lighting or color grading in assessing video similarity.

In the **ColorGrading** test, the perceived visual difference caused by the color grading is evaluated by measuring, for each frame, the CIELab distance and taking the mean as a global value. Since the camera paths are set such that the objects observed are more or less always the same in all the frames this measure is a reasonable choice to objectively evaluating the impact of the color grading for each video. Taking this distance into account, we obtain a Kendall τ-c of -0.4045 (with high significance) and a Spearman's ρ of -0.8 w.r.t to the subjective evaluation. Instead, the optical flow predominates over color stimulus with a strong correlation with the human judgements (Kendall's $\tau-\mathrm{c} = 0.4045$, Spearman $\rho = 0.8$). These values are exactly the opposite because the videos are generated such that the color grading distance is inversely correlated with the ϕ distance.

We obtained very similar results for the **Lighting** test. Videos produced under different lighting conditions were perceived as similar to the reference one thanks to the low differences in the optical flow, that is low values of ϕ. Taking into account that the use of a different time for the daylight simulation leads to very different shadowing effects between the reference video and the videos rated as the most similar, this again demonstrates the importance of the optical flow in evaluating the perceived visual similarity.

2.8.2 Test Data and Performance

In order to test our approach, we downloaded several models from public repositories (Trimble 3D Warehouse, Archive3D). Table 2.2 shows a view of these models along with statistics regarding their size, time required to build the database of instant flows, and the corresponding database size. We performed the flow sampling using 16 orientations and 18 directions of displacement. The first layer of our sampling was done inside the bounding box of the scene inflated by a factor 0.3. Inflating the bounding box

	 Museum	 Sibenik	 Town	 Floorplan	 House 1	 House 2	 House 3
Input							
#polygons	1,468,260	69,853	14,865	5,684,739	38,367	22,026	32,051
Database							
# Instant Flows	3,952,512	2,996,064	2,453,184	3,743,712	1,506,240	2,282,688	1,991,232
Size (GB)	1.37	1.03	0.87	1.29	0.53	0.81	0.70
Proc. Time							
Sampling	2.03	0.19	1.07	7.25	0.12	0.16	0.19
Instant Flows	5.15	3.41	3.01	5.4	1.52	2.45	2.27

Table 2.2: *Statistics on the scene pre-processing for the 3D models used in our experiments (timing in hours). Note the compactness of the databases despite the large number of instants flows stored. Processing times are acceptable given the fact that datasets are built only once and can be reused for any video.*

allows the cameras to be positioned outside the scene as well. As shown in Table2.3, the size of the database is not related to the size of the scene as much to its complexity. This is not surprising, since the more complex the scene the more chances there are for different flows to be create by the sampling. With respect to the database sizes and calculation times, despite the high number of instant flows stored, the databases have a manageable size and their creation time is acceptable given the fact that they are created only once and can be reused for any video.

Video Sequences								
Name	Length	Track&Flow	Query - Step HM	Query - Step EMD	Candidates Paths	Clustering	Final Refinement	Total
Tree	1.33	30.15	3.23	154.98	12.77	1.81	116.23	319.17
Showreel	3.84	84.25	14.25	392.54	55.98	4.55	272.17	823.74
Documentary	5.07	104.21	16.05	452.40	71.30	3.22	442.29	1,089.47
Ambulance	5.07	104.13	14.23	530.25	66.23	2.92	369.21	1,086.97
Villa	20.75	417.23	83.41	1,846.80	296.07	26.14	1,813.69	4,483.34
GreatBeauty	17.07	363.87	72.02	1,252.68	224.38	13.25	1,459.44	3,385.64
House1	17.57	n/a	24.75	1,448.43	143.47	15.21	1,179.21	2,811.07
House2	17.57	n/a	28.35	1,448.43	121.55	18.80	1,165.81	2,782.86
House3	17.57	n/a	26.21	1,448.43	156.22	17.51	1,181.97	2,830.34

Table 2.3: *Processing times for the video sequences (timing in seconds). The table also shows the performance details for the main stages of our algorithm. For the videos House1, House2 and House3 there is no camera track estimation nor optical flow calculation as the input is a known virtual camera path designed on a different CAD model.*

The list of input sequences used in our tests is given in Table 2.3. The corresponding results are shown in the accompanying video. The first five entries of the table are various shots taken from YouTube. “The Great Beauty" is the initial sequence of shots of the famous Oscar winning movie by Paolo Sorrentino. Using this kind of movie is an entertaining use of our system. The most practical use is to automatically produce presentations for a portfolio of 3D models. To demonstrate this application, we took three shots of a simple CAD model and then replicated them automatically for three different models (referred to as “House1”, “House2” and “House3” in Table 2.3).

Table 2.3 also reports the overall processing time for each video and the performances of the main stages of our pipeline. The “Track&Flow” column reports the time needed for the optical flow and the camera path estimations. The “Query - Step HM” and “Query - Step EMD” columns provide details on our two-step query approach (as described in Section 2.6). Finally, the “Candidates Paths”, “Clustering” and “Final

Refinement" columns report the processing times for the candidates paths selection, the clustering and the NEWUOA optimization, respectively. The total processing time is reasonable and is dominated by the EMD re-ranking and by the final optimization step. We used 100 clusters with the final refinement applied to the 5 best sequences. NEWUOA optimization is one of the most time-consuming step of our pipeline, however, in most of our tests, this optimization does not significantly improve the final quality. All tests were run on a desktop PC equipped with an Intel Core I7-4820K CPU and an nVidia GeForce GTX 780 GPU.

Finally, for the sake of completeness, we show two examples (see Figure 2.6 and Figure 2.7) of input videos and the corresponding output videos as a collection of frames. For a clear evaluation of the quality of the results obtained we refer to the accompanying video.

2.8.3 Discussion and Limitations

To the best of our knowledge, there are no other approaches that use a video guidance to produce automatic navigation of a scene. So, we give here a qualitative comparison w.r.t other methods instead of a quantitative comparison. Many approaches share broadly the same scheme, i.e.: computing a network of non colliding paths using distance field-like approaches and use them to constrain the camera movement. Their final goal is to control the camera to show the scene as in [3, 28, 101] or to follow a specific target inside a scene [92]. Avoiding collisions and, more generally, keeping the camera away from the scene surface are limitations our approach does not have. In fact, if the input video is a sequence going through a wall, the computed camera path will most likely do the same. By not precomputing paths, but only a dense sampling of instant flows, we have much more flexibility in the camera paths that can be created, and providing an example video is just a way to tell the system what type of path to create. On the other hand, our approach does not guarantee that every part of the scene will be shown in an output video, or that a specific landmark will be seen (although it could be easily constraint the search space to instant flows in the region around a specific region of interest).

The main limitation of our approach is the inability to work with dynamic scenes, where animated characters move and interact. In such cases, it is very difficult to try to obtain a similar flow field in the output sequence. One possible solution is segmenting the computed optical flow (as in [82]) in order to isolate and hence ignore the moving foreground parts, thus limiting the influence of the other moving objects. Another issue may arise from the processing of the input video; the optical flow may be difficult to estimate reliably in the case of large textureless areas. For the same reason, estimating the camera path may fail or provide poor results. Finally, the intrinsic camera parameters are defined at scene sampling time. While this limitation may be overridden by extending the sampling to cover a range of intrinsic parameters, the dynamic combination of camera movement and focal change would add further uncertainty in terms of camera tracking and finding candidate solutions.

Figure 2.6: *Output video (2nd row) produced using the 3D model Sibenik and the input video shown in the first row. The algorithm is able to reproduce the moment of passage of the tree by replacing it with a column of the 3D model.*

Figure 2.7: *Output video (2nd row) produced using the 3D model Museum and the input video shown in the first row. The algorithm maps a video of a camera entering a door to a path where the virtual camera enters an arch of the 3D model.*

CHAPTER 3

3D Presentation from Images: Stereoscopic Navigation

3.1 Introduction

In this chapter [1], we deal with the problem of presenting a 3D scene taken from the real world when its geometry is not available and can not be adequately reconstructed by automatic image-based methods. As introduced in Section 1.2.2, our method takes in input a set of calibrated images and exploits their spatial organization to build a graph of the suitable stereo-pairs. In this graph, we associate each node with a calibrated camera that represents a virtual eye. Two virtual eyes give a stereo pair. Along each edge of this graph, we can instantiate a new camera using simple linear interpolation of the extrinsic parameters. We can then generate a new image of the scene, as seen by the new camera, by using known IBR techniques. In this way, we extend the set of the possible views from the discrete set of acquired cameras to a continuous domain given by our graph. Combining any couple of cameras that we can pick on this graph we obtain the set of all possible stereo pairs. This set, which we named *StereoSpace*, can be then navigated by the user via simple pan/zoom interactions using the mouse interface. Note that not all the stereo pairs that are optically correct are also comfortable to the user. Other factors, such as depth complexity and direction of movement, influence the perception of a 3D view and there are metric to assess the comfort of a stereo pair. We will show how our stereoscopic navigation with the StereoSpace can integrate any metric to exclude uncomfortable regions from the navigation. In summary, this chapter makes the following contributions:

[1] A preliminary version of the work described in this chapter was presented in [9]

- A formal definition of StereoSpace as the space of the stereo pairs which we generate interpolating the actual viewpoints given a set of calibrated cameras.
- A stereoscopic navigation paradigm based on an intuitive pan/zoom interaction that enables the user to inspect the scene and allows the integration of comfort metrics.
- A visualization application supporting stereoscopic devices of any type (e.g. shutter glasses, anaglyphs, etc.) that provides an immersive experience without the need of a complete 3D reconstruction of the location/object of interest.

To better appreciate the results of this chapter, we provide an accompanying video at the address `http://www.andreabaldacci.it/publications/#stereonav`.

3.2 Related Work

In the following, we review the literature most closely related to our approach.

Image Based Rendering The literature on IBR is vast. Levoy et al. [81] proposed one of the first approaches, the so called Light Field Rendering. In this method, they generate a new view by re-sampling a large set of images of the scene of interest. If the original set is dense enough, the synthesized view has a very high-quality, otherwise ghosting or blurring artifacts are visible. Afterward, several researchers proposed solutions to enhance the rendering quality using only sparsely sampled image sets. These methods are based on a proxy geometry, i.e. a coarse approximation of the real underlying geometry of the acquired object. Outstanding examples are Lumigraph rendering [51], Unstructured Lumigraph rendering [19] and View-dependent Texture Mapping (VDTM) [27]. Possible misalignments due to the coarseness of the proxy or small errors in cameras calibration can produce ghosting artifacts. Eisemann et al. [35], for example, suggest to correct such artifacts by warping the projected images using optical flow. In general, reconstructed proxies may miss entire regions of the corresponding images thus leading to poor rendering quality. Goesele et al. propose a workaround to this problem with the Ambient point clouds method [48] which use non-photorealistic rendering to render the transition between images in poorly reconstructed or missing regions. The most recent approaches to IBR are based on variational warping methods. In these methods, image warping builds on the sparse correspondences given by the projection of 3D points generated by any structure-from-motion system. In particular, Zhou et al. [130] used this method for 3D video stabilization while, in [23], Chaurasia et al. used the same approach for wide-baseline IBR. The essence of these methods is to lay down a regular grid or triangulation over each picture. Each vertex of this grid becomes an unknown to be computed in its warped version in the final image. We can the impose two energy terms: a data term and a similarity term [130]. In the data term, the recovered sparse 3D points are projected onto the grid, and their coordinates are expressed by barycentric coordinates or by the bi-linear interpolation of their four surrounding vertices. Then, the squared distance between the interpolated grid position in the output grid and the projected 3D point in the novel image is minimized. A similarity constraint is included to reduce local shape distortions of each grid triangle which thus will undergo a transformation as close as possible to a similarity. In the

wide-baseline case, attention must be taken to depth discontinuities which lead to unnatural silhouettes distortions [23]. In [23], Chaurasia et al. overcome this problem by requiring manual silhouettes annotation and including an ad-hoc energy component which affects only the edges along silhouettes. In [22], Chaurasia et al. avoid the manual intervention by employing a local warping approach which deforms a superpixel segmentation of the images, warping each super-pixel separately.

2D-to-3D conversion In a way, our work is comparable to the 2D-to-3D conversions technologies used in movies post-productions. The vast majority of these works, however, concern stereoscopic view generation with small baseline (e.g. video). Knorr et al. [70, 71] propose a system for the production of a 3D stereoscopic video from a monocular one. They first use a structure from motion system to recover both camera motion and a sparse 3D point cloud of the filmed scene. After selecting a camera of the sequence, they instantiate a virtual camera applying a horizontal offset to it, thus creating a virtual stereo rig. The 3D point cloud is then projected onto the new view and to nearby cameras. The projected points correspondences between the new view and the other images are then used to calculate a homography transformation for each image. Each neighboring view is then warped into the virtual stereo frame and blended to form the final image. This approach makes the assumption of having a small baseline between cameras and, more radically, it approximates the filmed scene to a planar scene. An alternative technique, named "Depth-image-based rendering" (DIBR), is frequently used in movies post-production and it consists in creating a dense depth map for each monocular image. The depth map creation process can be either the accurate but manual process of an artist or the automatic creation of "surrogate depth maps" using simple 2D features such as luminance intensity, color distribution or edges [34]. These methods do not recover the real depth of the scene. Instead, they create a depth which is only approximatively consistent but provides a comfortable user experience. The depth map is then converted to a disparity map and the original image is warped accordingly to obtain the new view [37, 76].

3.3 Method Overview

The key observation behind our method is that a set of pictures taken to perform a 3D reconstruction is well suited to create stereo views because of following characteristics (see Figure 3.1 for practical example):

- The cameras tend to focus on the same region.
- Their positions are arranged on arcs surrounding the object.
- The height of each camera is approximatively the same (that is, the height of the person who is holding the camera).

Given these features, it is very likely that a good number of camera pairs can be rectified to create a stereo pair. Our approach consists of extending the domain of views from the original set of cameras to a continuous space so that the space of the achievable stereo pairs is also continuous. We call this domain *StereoSpace* and we will show how it can be used to provide a seamless stereoscopic browsing experience.

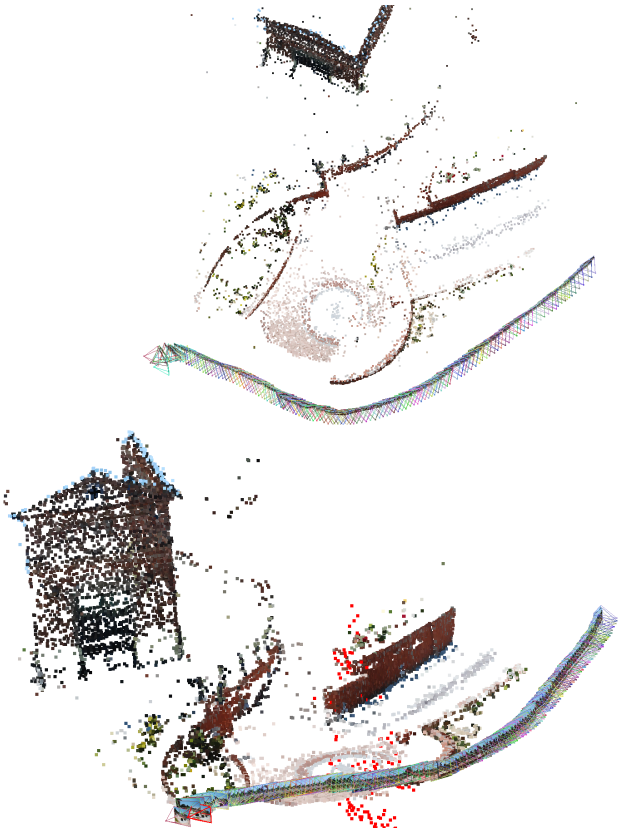


Figure 3.1: *A typical acquisition pattern for a building.*

In the following, we illustrate how we build a graph of the suitable stereo pairs and how image interpolation and rectification are achieved. In Sections 3.4 and 3.4.1, we better formalize the concept of *StereoSpace* and how to map it on the proposed navigation system. In Section 3.5, we show how to integrate into our system a stereoscopic comfort measure to improve the user experience. Finally, we present some results in Section 3.6.

We start by processing the input images to obtain a set of calibrated cameras and a sparse point cloud of the scene. For this first step, we use VisualSFM [121, 123].

For image interpolation, we use the state of the art IBR engine presented in [22] by Chaurasia et al. This algorithm requires an offline preprocessing phase for superpixel segmentation and depth completion that takes in input the calibrated cameras and the point cloud as calculated in the first step.

Once two novel views are generated, we have to rectify them to simulate a horizontal stereo rig. To accomplish this task, we used the rectification algorithm by Fusiello et al [45] which is a simple and straightforward linear method. In summary, this method uses a standard pinhole camera modeled by its center C and its image plane I located at a distance f from C , where f is the *focal length* of the camera. Cameras are already calibrated thus we have full knowledge of their *perspective projection camera*. Given two cameras whose centers are $\mathbf{C_0}$ and $\mathbf{C_1}$ in homogeneous coordinates, and whose camera matrices are $\mathbf{M_0}$ and $\mathbf{M_1}$, the idea behind the rectification process is to define two novel matrices $\mathbf{M_0^1}$ and $\mathbf{M_1^1}$ which preserve their previous points of view but they

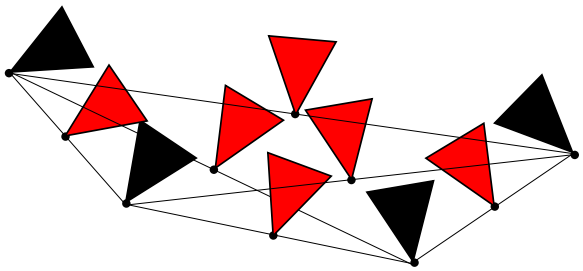


Figure 3.2: *Four cameras (in black) and the corresponding six stereo cameras (in red).*

are rotated to make the two image planes coplanar and parallel to the baseline C_0C_1 thus ensuring that also the epipolar lines are both parallel and horizontal. Moreover, forcing the intrinsic parameters of the cameras to be equal, conjugate points lay on the same line in both images having the same vertical coordinates.

To build the initial graph of the suitable stereo pairs, we first connect every possible pair of cameras. We then prune the graph to retain only the pairs that are already reasonably close to a stereo rig and thus can be rectified successfully. We first sort all edges by length, we then iteratively remove the longest ones as long as the graph connectivity is preserved. We then filter out all the remaining connections using two criteria. In the first, we check if the angle between the cameras view directions is more than $\theta = 30°$. In the second one, we check if the cameras are reasonably aligned by measuring the angle β between the vector connecting their positions and their x-axis. If $\beta > 30°$ we disconnect the pair.

3.4 StereoSpace: the domain of stereo views

A stereo pair is defined by a *view position*, that is the midpoint between the camera pair, an *interpupillary distance*, that is the distance between the camera pair, and a *view orientation*, which is obtained as shown in the previous section. Here we want to define a browsing space where the user can change the point of view and the inter-pupillary distance. We can thus identify a stereo pair just from the positions of the camera pair. Therefore each camera pair maps to a stereo view as:

$$s(a, b) = ((a + b)/2, \|a - b\|) \tag{3.1}$$

where $a, b \in \mathbf{R}^3$ are the camera positions and $s \in \mathbf{R}^4$ is the position of the stereo camera enriched with the interpupillary distance. Let be $\mathcal{C}$ the set of camera positions, we define the *SteroSpace* as the codomain of function s:

$$S(\mathcal{C}) = \{s(a, b)|a, b \in \mathcal{C}\} \tag{3.2}$$

In this simplest setting, the resulting StereoSpace is a set of points. Figure 3.2 shows a simple example where 4 cameras (in black) are paired (connected by a segment in the figure) to make 6 stereo cameras (in red). In this case, the only browsing modality for the user would be to jump from one fixed stereo view to another.
By applying camera interpolation as described in the previous section, the space of cameras is extended from a set of point to a set of segments $=\{s_0, \ldots, s_m\}$ where each segment $s = \overline{ab}$ connects two cameras of the original dataset (we can include

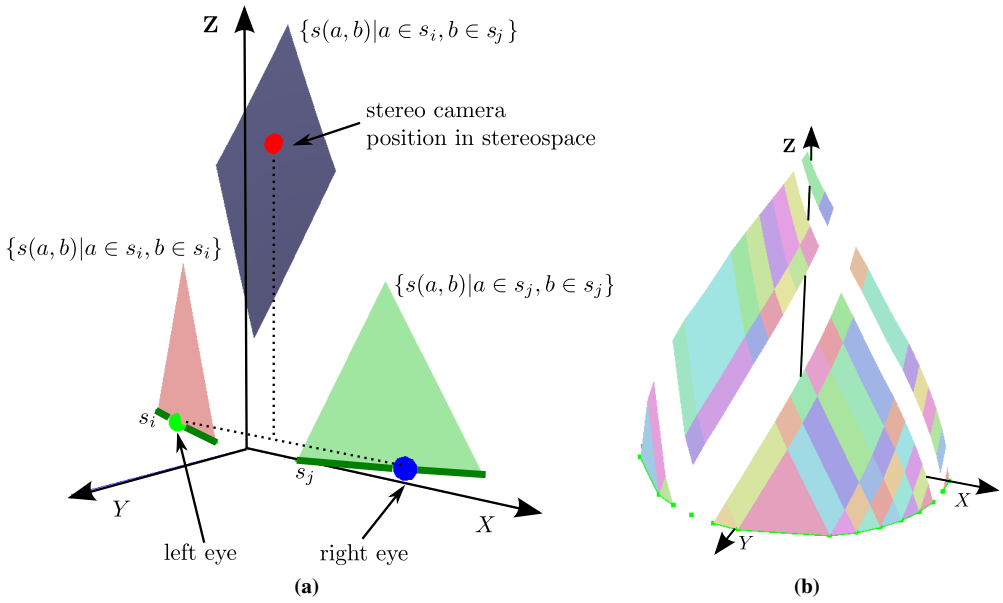


Figure 3.3: *The StereoSpace for two segments and for polylines. For illustration purposes we make the simplification assumption that the original cameras lay in a common plane.*

in $\mathcal{C}$ the original camera positions by considering degenerate segments $s = \overline{aa}$). In this case, the corresponding StereoSpace become piecewise continuous. Let us start by considering the StereoSpace corresponding to a pair of segments s_i and s_j (please refer to Figure 3.3a). As a simplification assumption (that can be removed later) and for illustration purposes, let us assume that the original cameras are in a common plane, say XZ, so that we can reduce the dimension by one and map the interpupillary distance on the Z axis. The position of the stereo camera is mapped on the XY plane (because it is the halfway point between two points in that plane) and the interpupillary distance on the Z axis. The resulting StereoSpace is the union of three continuous regions:

$$\begin{aligned} S(\{s_i, s_j\}) = &\{s(a,b)|a \in s_i, b \in s_j\} \cup \\ &\{s(a,b)|a,b \in s_i\} \cup \\ &\{s(a,b)|a,b \in s_j\} \end{aligned} \tag{3.3}$$

The portion of StereoSpace generated by a pair of points belonging to different segments is a rhomboid-like shape, while if they belong to the same segment, the regions are triangular (which are folded rhombi). Note that the projection on the XY plane of these regions is not other the Minkowski sum of the segments obtained by halving the coordinates of s_i and s_j.

Figure 3.3b shows the StereoSpace for a set of segments connected to form a sequence of polylines. This is a typical situation we have for cameras following a reconstruction-driven pattern. For the sake of illustration, we used different colors for subregions generated by different segments within the same polyline.

3.4.1 Stereo Browsing with the StereoSpace

One way to move inside the StereoSpace would be just to visualize its geometric representation in a separate window, let the user click on a point and move the view to the corresponding stereo camera. However, we can provide a much more natural interaction. Let p be the current viewer position in the StereoSpace. It is clear by definition that we move from point p to a point with a greater z component we will increase the interpupillary distance, which corresponds to scaling down the model and bringing it closer to the viewer, something that we improperly call zoom-in in our interface. Conversely, moving to a point with lower z-value means to decrease the interpupillary distance, that is, scaling up and bringing the model farther away (zoom-out).
If we move to a point in the same XY plane (that is, leaving the z component unchanged), there will be no zooming involved. In the typical configuration of camera positions, this horizontal movement will correspond to a left-right pan or a horizontal orbit. $\mathcal{C}$ can be easily parameterized with the index of the segment i and the linear interpolation coefficient between the endpoints λ_i:

$$c(i, \lambda_i) = s_{i0}\ (1 - \lambda_i) + s_{i1}\ \lambda_i : i \in [0 \dots m], 0 \leq \lambda_i \leq 1 \tag{3.4}$$

where $\overline{s_{i0}\ s_{i1}}$ is the segment i (we will just indicate (i, λ_i) with λ_i from now on). A point in StereoSpace is then defined as:

$$s(\lambda_i, \lambda_j) = \begin{bmatrix} \frac{1}{2}(c(\lambda_i) + c(\lambda_j)) \\ \|c(\lambda_i) - c(\lambda_j)\| \end{bmatrix} \tag{3.5}$$

The gradient of $\nabla s(\lambda_i, \lambda_j)_z$ tells us the direction of maximum increment of p_z (for the zoom-int/zoom-out movement), while the tangential direction $T(\lambda_i, \lambda_j) : T \cdot \nabla s(\lambda_i, \lambda_j)_z = 0$ is the direction where no zoom takes place (see Figure 3.4a). Thus we can map the user commands to movements in parametric space as:

$$\begin{aligned} zoom(v) &\rightarrow [\lambda_i, \lambda_j]^T += v\ \nabla s(\lambda_i, \lambda_j)_z \\ pan(v) &\rightarrow [\lambda_i, \lambda_j]^T += v\ T \end{aligned} \tag{3.6}$$

where v is the amount of movement (positive or negative). In this way, the user can move with two degrees of freedom and the position is updated to the *best* position in StereoSpace. With this mapping, we provided the way to smoothly change position and zoom within a continuous region of the StereoSpace. As shown in Figure 3.4b, when the current position is on the border of a region of the StereoSpace, we jump to a neighbor region in the direction of movement. In other words, if the user is zooming-in, that is increasing the z component in StereoSpace, we will look if there is a region of the stereo space above the current one and so on. Please note that this time the direction is expressed in StereoSpace and it is the mapping of the moving direction in parametric space. That is:

$$d = s(\lambda_{dir}) \tag{3.7}$$

We jump to the closer point on the StereoSpace which is in the cone with apex in the current position, oriented as d and with angle $40°$. In our implementation, this is done

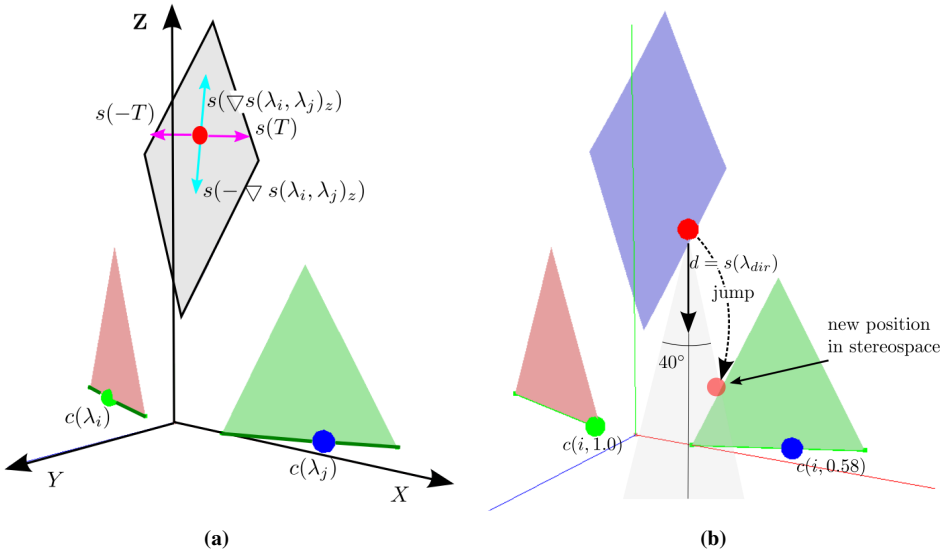


Figure 3.4: *Moving in the StereoSpace. Figure (a) shows the pan and zoom directions mapped to the StereoSpace. In Figure (b), we show our strategy for jumping between StereoSpace regions.*

by creating tessellated surfaces for each region and inserting them into a search data structure. When we pick a particular position in StereoSpace, we have the problem of finding its projection in parametric space, that is, the λs. This situation happens because we have to jump from one point to another of the StereoSpace directly, instead of changing the position in parametric space.

3.4.2 From StereoSpace to parametric space

The function $s : \mathcal{C} \times \mathcal{C} \rightarrow S$ is not injective so there is not s^{-1}. However, we can still find a mapping from StereoSpace to parametric space. There are two cases in which a point in StereoSpace may correspond to more than one point in parametric space.
The first case is because two regions, corresponding to difference couples of segments,

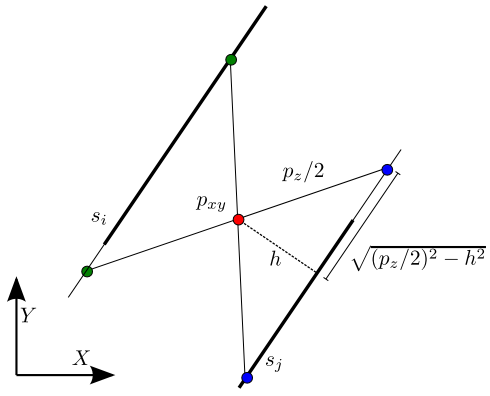


Figure 3.5: *Inverse mapping from the StereoSpace back to parametric space in the case of parallel segments.*

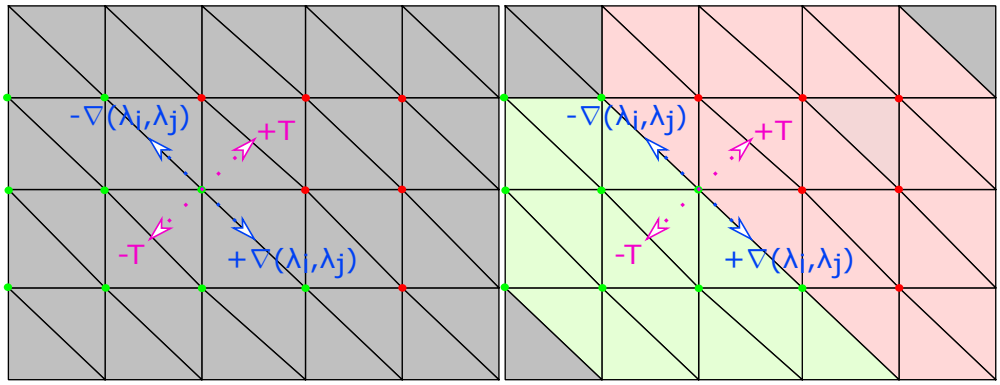


Figure 3.6: *Sampling stereoscopic comfort in the StereoSpace. In Figure (a) we show the sampling directions and the visited vertices marked in green or red if the corresponding stereo pair is comfortable or not. In Figure (b), the one ring neighborhood of uncomfortable vertices is marked (in red) for removal.*

intersect each other. When we jump from a position to a point which lies at this intersection, we just choose to map in the first region we find. This is easily done by storing in the geometric representation of the region, two references to the segments that generate it. The second case is when a region if produced by two parallel segments, see Figure 3.5. This case happens for all the triangular regions provided by a single segment (that is, by two instances of it). Let us consider the point p in the region region generated by s_i and s_j:

$$\begin{aligned} p_{xy}(\lambda_i, \lambda_j) =& s_{i0_{xy}} (1 - \lambda_i) + s_{i1_{xy}} \lambda_i + \\ & s_{j0_{xy}} (1 - \lambda_j) + s_{j1_{xy}} \lambda_j \end{aligned} \tag{3.8}$$

solving for λ_i, λ_j leads to a simple system:

$$\begin{bmatrix} s_{i1_{xy}} - s_{i0_{xy}} & | & s_{j1_{xy}} - s_{j0_{xy}} \end{bmatrix} \begin{bmatrix} \lambda_i \\ \lambda_j \end{bmatrix} = p_{xy} - \begin{bmatrix} s_{i0_{xy}} + s_{j0_{xy}} \end{bmatrix} \tag{3.9}$$

If the matrix is nonsingular, we can solve the system and have the λs, otherwise, we are in the case where the two segments are parallel. In this case, we consider the projection of p on the plane XY and then look for points on the segments at a distance $p_z/2$ from p_{xy}, which can be easily done by computing the distance from each segment and using the Pythagorean theorem. Note that we may have two solutions from which we will choose the most coherent with the rectified camera orientation.

3.5 Navigation in comfort zone

As mentioned in the introduction, the user experience may vary significantly in viewing stereoscopic images even when a stereo pair is geometrically correct. A simple example consists of camera pairs where the view directions are too much convergent.

In this case, the (disturbing) user experience would be to see the scene on the tip of his/her nose. We briefly review the motivations behind users discomfort in viewing stereoscopic images and then we show how to circumvent such problem in our system.

Background in stereoscopic vision Depth perception in the human visual system is achieved using several cues. Some of these cues are provided independently by each eye, i.e. they are monocular, for example: accommodation, motion parallax, focus/defocus and linear perspective. Conversely, vergence (also called convergence) and stereopsis are binocular as they depend on the interplay between the two eyes and the images formed onto each retina [60]. Vergence is essentially a triangulation system. The depth evaluation depends on our inter-pupillary distance and on the point in space where the lines of sight meet, and thus it depends on the angle they form [115]. To ensure that the two lines of sight meet at the same point in space, our visual system solves a stereo matching problem i.e. it tries to find corresponding points between the two retinal images. Another process which implies the comparison of the two retinal images is disparity which analyzes the offset between corresponding points on the left and right eyes. The difference between these two visual mechanisms is that disparity does not depend on the directions of the two lines of sight. The disparity is a relative depth cue as it measures depth relative to the *horopter* that is the locus of points for which no disparity is perceived (i.e. points on the horopter project on the same positions in both eyes). The *horopter* is a circle passing through the centers of the two eyes and their fixation-point. We have two types of disparities: crossed disparity and the uncrossed one. Objects appear nearer to the subject when their corresponding points meet in front of the fixation point thus generating a crossed disparity. Instead, in the uncrossed disparity, corresponding points meet behind the fixation point thus appearing to be farther. Our visual system can adequately fuse the left and right images only inside a particular disparity range around the horopter called Panum's fusional area. Disparity values outside this range generate double vision. Moreover, not all disparity values inside the Panum's range can be fused comfortably [77]. In fact, applications like 3DTV or 3D cinema try to limit the range of disparities to a third of the whole fusible range, the so-called *Parceval's zone of comfort* [78]. There are many other concerns about the current generation of stereoscopic vision technologies. For example, in current theaters, spectators can have conflicting depth cues as they focus their eyes at the screen distance, while the images disparity may suggest a different depth complexity of the scene. This problem is usually referred as eyes accommodation/disparity conflict [57, 61]. For a complete list of discomfort factors, we refer to [97].

We defined the StereoSpace as the region of space from which we can build a stereo pair. However, not all stereo pairs are alike. The quality of a stereo pair is related to several factors: the quality of the interpolation between cameras, the quality of the rectification and, as just stated, the capability of the user to fuse the two views in a comfortable way. We can seamlessly incorporate any comfort measure in our navigation paradigm by performing a preprocessing phase where the whole StereoSpace is densely sampled and every possible stereo pair is evaluated. Then *uncomfortable* zones are then marked and the navigation algorithm is modified to avoid them without jagged movements. Figure 3.6a shows a tessellated portion of the StereoSpace parametric space, obtained offline by triangulating a grid with fixed step for parameters λ_i

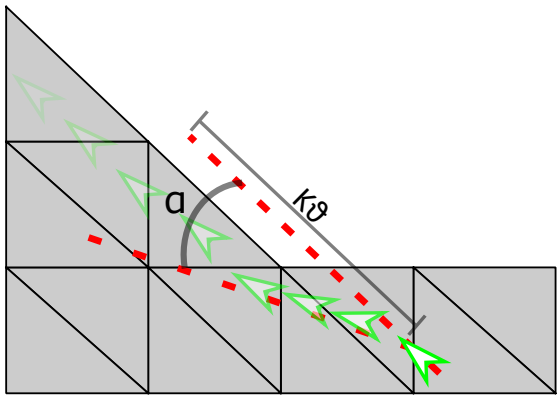


Figure 3.7: *Schematic representation of the steering algorithm to avoid uncomfortable zones. The green arrow indicates the current position in the parametric space.*

and λ_j. We recall that each point in the parametric space (and its projection in the StereoSpace) corresponds to a particular stereo pair. We thus evaluate a comfort measure over all vertices and we mark them appropriately. Several comfort metrics exist in literature [29, 30, 33, 66, 128]. Some of these metrics evaluate the comfort of a single stereo pair while others are devised to assess the comfort of short motion sequences. As our navigation interface allows to move the viewpoint seamlessly and not just jumping from a static image to another, we chose the state of the art comfort metric by Du et al. [33]. This metric is conceived for stereo animations and hence it does not concern just a single stereo pair but a sequence of them. As shown in Figure 3.6a, for each vertex, we generate four distinct sequences of stereo frames for each movement direction and for each verse (i.e. pan left/right and zoom in/out). We then apply the comfort measure in [33] to each sequence. If at least one sequence is not comfortable, we conservatively decide to exclude the current vertex and all its adjacent faces from the comfort zone (see Figure 3.6b).

We modify the navigation algorithm presented in Section 3.4.1 to avoid uncomfortable zones. Our goal is to gently deflect navigation paths that would cross uncomfortable zones without abrupt changes of direction. We recall from Eq. 3.6 and Eq. 3.7 that λ_{dir} and v are the direction and amount of displacement, respectively. Now, let $\lambda = (\lambda_i, \lambda_j)$ the current stereo pair, in parametric space. At every user interaction, we trace a ray from the current position λ in the λ_{dir} and we perform a simple ray-triangle intersection test to check if the current direction of movement crosses any uncomfortable zone at a distance less than kv. If this happens, we look for the smallest deviation from λ_{dir} that does not cross any uncomfortable zone within a cone of directions of α degrees, and we move in that direction instead (we empirically fine tuned k to 10 and α to $60°$). See Figure 3.7 for a schematic representation. Clearly, as the steering cone is narrow, the user could find himself stuck along the border of a large region. As we do not want to deflect too much from the original user intention, if it is not possible to avoid an uncomfortable zone, we jump across it by repositioning in the parametric space to the nearest comfortable point in the direction of movement (we demonstrate such case in Figure 3.8). If such point does not exist, we jump to the closest region of the StereoSpace (i.e. another group of cameras) in the direction of movement as described in Section 3.4.1.

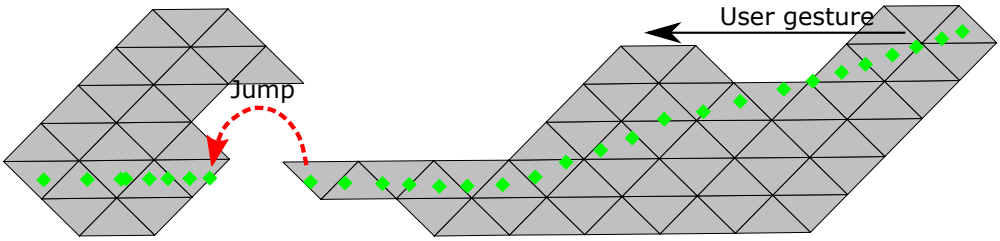


Figure 3.8: *The registered navigation session shows how our algorithm steers from the uncomfortable region. At some point the system decides to jump as the border of the hole cannot to be avoided.*

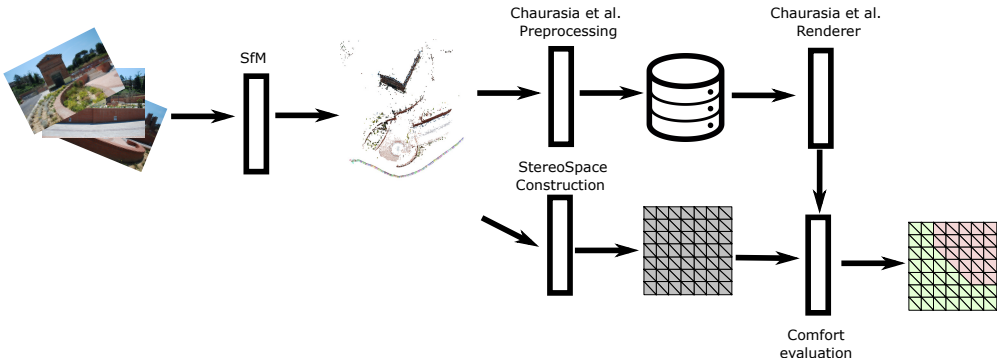


Figure 3.9: *Our data processing pipeline*

3.6 Results

We now illustrate the application based on the StereoSpace navigation. Figure 3.9 shows a scheme of the data processing occurring in our application. The input to our application is a set of images. The first step consists of calibrating the cameras and generating a sparse point cloud of the scene. We then run the offline process described in [22] to prepare the data structure for the on-the-fly synthesis of interpolated frames. At this point, we generate the cameras graph with the associated StereoSpace and, after tessellating the corresponding parametric space, the comfort measure [33] is computed and the uncomfortable zones are marked as such and will be avoided in the navigation.

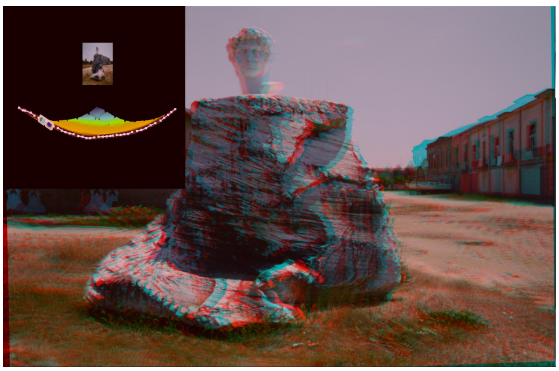


Figure 3.10: *An image of our prototype stereo browsing system.*

In Figure 3.10, we show the interface of our application. An automatically generated navigation map which can be activated/deactivated is drawn at the top-left of the main window. The map is provided just to improve the sense of orientation of the user since there is not any direct user interaction with it. The acquired scene is represented with an image as an impostor for its real geometry. The polylines connecting the calibrated cameras are drawn to show the camera path around the subject/location of interest. The location of the stereo camera and both left and right eyes are instead represented respectively by a small image of anaglyph glasses and with a colored sphere for each eye. To visualize the StereoSpace, we render it with an orthogonal camera from above, with a color per vertex corresponding to the comfort for that point.We have tested our application on several scenes of various detail and complexity. From Figure 3.11 to Figure 3.14, we show some sequences where the user was panning or zooming. In all images, left and right eyes are both interpolated images.

3.6.1 Discussion & Limitations

The resulting images are capable of giving a strong sense of depth as can be seen, for example, in Figure 3.11. We show a particularly challenging scene mainly made up of textureless surfaces in Figure 3.13. As opposed to multi-view stereo methods, our system is robust to these situations as the input point cloud is used only to warp the input images. As can be seen in the "Camion" sequence of the accompanying video, some points oscillate between crossed and uncrossed disparities as the camera moves. This behavior is correct as the camera is not moving at a constant distance with respect to the scene. Because of this, points which are close to the boundary of the horopter cross the point of zero disparity several times as the camera moves. As we rectify cameras whose view directions are not parallel, we only approximate a real stereo rig and thus some slight disparity variation may occur in unwanted situations. From our tests, however, the viewer seems to tolerate well this problem. As we generate the left and right images of our simulated stereo rig independently, some slight artifacts can be present in one eye but not in the other, leading to mismatched stimuli. From our tests, this does not result in a significant binocular rivalry as the artifacts are small and thus seems to be quite tolerable for the viewer (as already noted in [78]).

Figure 3.11: *A panning sequence of the Museum scene. This dataset gives a strong sense of depth of the arcade and the space beneath it.*

Figure 3.12: *A panning sequence of the University scene.*

Figure 3.13: *A panning sequence of the Puppets scene. For multi-view stereo methods, this scene is especially challenging as its surfaces do not have detailed textures. This leads to a very noisy and incomplete reconstruction. Our method, however, is robust to these situations as the point cloud is used only to warp the input images.*

Figure 3.14: *A zooming sequence of the Camion scene. Here, the user can easily adjust the stereoscopic effect as long as it remains in the comfort zone. Note how the tree in the background starts from an almost flat appearance to a final one where is detached from the wall behind.*

CHAPTER 4

3D Presentation from Images: Geometry Reconstruction

4.1 Introduction

In this chapter [1], we present a novel interactive framework for improving 3D reconstruction starting from incomplete or noisy results obtained through automatic image-based reconstruction algorithms. As introduced in Section 1.2.3, the core idea is to enable the user to provide localized hints on the curvature of the surface, which are turned into constraints during an energy minimization reconstruction. To make this task simple, we propose two algorithms. The first is a multi-view segmentation algorithm that allows the user to propagate the foreground selection of one or more images both to all the images of the input set and to the 3D points, to accurately select the part of the scene to be reconstructed. The second is a fast GPU-based algorithm for the reconstruction of smooth surfaces from multiple views, which incorporates the hints provided by the user. We show that our framework can turn a poor-quality reconstruction produced with state of the art image-based reconstruction methods into a high-quality one.

In summary, this chapter makes the following contributions:

- A multi-view segmentation algorithm for the joint foreground/background segmentation of the input calibrated images and its associated sparse 3D point cloud.
- A smart user interface though which the user can easily add hints on a minimal number of input images.
- A GPU-friendly formulation of an energy minimization reconstruction.

[1] The work described in this chapter appeared in [8]

- A fast GPU-based multigrid solver for the above formulation.

To appreciate the results of this chapter, we provide an accompanying video at the address `http://www.andreabaldacci.it/publications/#recon`.

4.2 Related work

This chapter contributes to two different fields, both of which have a substantial body of literature: Image Segmentation and Multi-View Reconstruction. In the following we concisely review the state of the art on both fields, focusing on those algorithms that are more closely related to our work.

4.2.1 Image Segmentation

Image Segmentation refers to partitioning the pixels of an image into groups that share similar characteristics. Here, we are interested in partitioning the image into *foreground pixels*, which represent the object of interest, and *background pixels*. One of the earliest improvements on the crude manual segmentation were Intelligent Scissors [86], where the user defines various anchor points along the silhouette of the object, and a minimization algorithm adjusts the contour to match the gradient change of the image. Active Contours (or Snakes) [68] also work by minimizing an energy function over a contour (that is, a snake) accounting for image gradient and contour bending. More recent approaches do not work on the parametric definition of the contour but on the foreground/background classification of the pixels. In their seminal paper, Boykov and Jolly [14] recast the segmentation problem as a graph problem. More specifically, an image is mapped onto a graph where each pixel is a node and is connected to neighbor pixels by arcs. There are also two special terminal nodes, one for the background and one for the foreground, to which all other nodes are connected. The cost of a pixel-pixel edge is set to penalize separation between *similar* pixels, while the cost of a pixel-terminal penalizes setting the pixel to the background or foreground (for example on the basis of the initial manual pixel annotation by the user). With this formulation any *cut* in the graph corresponds to a partition of the pixels into two sets: those connected to the foreground terminal and those connected to the background terminal. The solution thus has a cost which is the sum of the cost of the arcs in the cut, which can be minimized by min-cut max-flow algorithms. Graph Cut methods have become the de-facto standard for image segmentation [2], thanks to their conceptual simplicity and to very efficient polynomial-time solvers [75]. Since their inception, the technique has been strengthened using shape prior information to disambiguate similar color areas [40], with an inclusion of Dijkstra's algorithm to preserve thin structures [117]. One of the most successful improvements is due to Rother et al. [98] who proposed an iterative algorithm where each iteration consists of solving the min-cut problem and re-assigning the cost functions on the basis of the solution found, until convergence. In addition, they introduced the Gaussian Mixture Model (GMM) to integrate color information in place of the simpler intensity histograms.

4.2.2 Multi-View Object Segmentation

Now that is common to have a set of calibrated images of an object, the problem of segmenting a single image has evolved into how several images (of the same scene) can be segmented at the same time. Graph-cuts can be naturally generalized to multiple images and thus many algorithms use them. All the approaches need to identify a way to connect pixels from different images. Sormann et al. [113] stack the images to form a 3D texture and use a preprocessing step to segment each image in clusters so as to reduce the size of the graph by using one node per cluster and not per-pixel. They assume a short baseline, that is, consecutive images in the stack do not change too much, so that the neighborhood among pixels of different images makes sense. Campbell et al. proposed the Volumetric Graph-Cuts [21], where the scene is voxelized and node-voxels are added to the graph, an idea somewhat reminiscent of the Voxel Coloring approach [106]. They use the *fixation* hypothesis, that is, that all the images look towards the object, which consequently is located roughly at the center of each image. They can thus simultaneously initialize the foreground/background and define the bounding box of the scene to be voxelized. The voxel nodes are essentially a means to connect pixels from different images. In a subsequent work [20] by the same authors the voxel grid was replaced by adding stereo correspondences and epipolar constraints to directly connect pixels from different images. Djelouah et al. [32] use a set of sample points uniformly distributed in the volume (under a similar hypothesis as in [21]) and consider the tuple of pixels defined by the projection of the sample point on the images. The key idea is that if all the elements of a tuple are classified as foreground, then the point belong to the object's surface. Similarly to [98], they use a GMM model and an iterative process for *a posteriori estimation* (MAP) of the classification variables. Sparse 3D samples are also used by Djelouah et al. [31], where the problem of multi-view segmentation is extended on the time dimension to support multi-view video segmentation and superpixels are used to reduce the computational complexity.

Other approaches, such as Bleyer et al. [13] and Kowdler et al. [1], assume that the objects in the scene can be approximated by planes, and that the baseline is so short that a reasonable depth map can be estimated. In this setting the segmentation can be set at an object level and 3D spatial relations between objects are used.

4.2.3 Multi-view Stereo Matching

According to [105] the multi-view dense stereo reconstruction algorithms can be categorized depending on their properties, for example depending on the surface representation used, on the reconstruction algorithm used, on the initial requirements, and initial hypotheses regarding the shape to reconstruct.

Many MVS reconstruction algorithms are based on segmentation. Typically, each image is segmented into the background and foreground (of the object of interest). One of the oldest of this class of methods is the *shape-from-silhouette*. Such methods estimate the *visual hull* of the object, that is the maximal surface consistent with the silhouette for all the views, by carving the volume of the object according to the silhouette in the different views. More recently, Yezzi and Soatto [126] explored the dual connection between the segmentation of an object in multiple images and 3D reconstruction of the underlying object. They employed a level set method, solved with

a multi-resolution scheme. Kolev et al. [74] reformulated the problem as a Bayesian estimation of the most probable shape that would yield the observed images, making the method more robust with respect to noise. In Sorman et al. [114] each image in the set is first clustered using mean-shift and then these clusters are segmented via GraphCut. However, segmentation happens sequentially, whereupon each segmentation provides a shape prior to be used in the subsequent one. Kolev et al. [72] deal with the image segmentation of all the views by projecting an evolving 3D surface. The problem is the setup in an energy minimization framework where the energy terms proposed take into account background and foreground terms plus a photo-consistency term. In a more recent work [73], the authors added an anisotropy term to this energy to also account for the orientation of the evolving surface. Jancosek et al. [63] compute an over-segmentation of the dataset as a first step in order to reduce the computational load and to provide priors for reconstructing flat areas of uniform colors. A recent hybrid (silhouette-based / correspondance-based) method capable to improve the reconstruction of objects with few visual features (e.g. uni-colored objects) has been proposed by Hoangminh et al. [91]. This method exploits the geometry reconstructed by means of standard SFM approaches to improve the automatic extraction of the silhouette. Another interesting paper to cite, even if not an MVS method, is the work of Liangliang et al. [90] which proposes a segmentation-based approach to complete the sparse reconstruction produced by scanned data.

Many other MVS algorithms work by estimating a depth map for each image and then integrating these depth maps into a unique surface. Goesele et al. [49] proposed a simple and effective algorithm to estimate the depth for each pixel by evaluating the photoconsistency (using NCC) of each 3D point estimated. Only pixels with high values of correlation are considered reliable. The sparse depth maps thus estimated are merged together by applying the volumetric surface reconstruction algorithm of Curless and Levoy [26]. Bradley et al. [15] developed a high-quality method by proposing a viewpoint adaptive window to drive the stereo matching between image pairs. The high quality depth maps thus generated are then merged together using a lower dimensional triangulation algorithm [50]. Furakawa et al. [43] proposed one of the most general and accurate algorithms for 3D reconstruction from calibrated images. This algorithm is the core of the PMVS software and is based on a patch representation of the surface. The initial oriented patches are estimated, then expanded to the nearby pixels, and filtered in an iterative way to produce a dense reconstruction.

Our approach is inspired by the multi-view segmentation based methods but uses depth maps as evolving surfaces to obtain the final reconstruction. A depth map is estimated for each camera by minimizing an energy functional composed of three terms: a smoothness term, a term to account for surface coherency which imposes that overlapping depth maps must coincide, and a term that takes into account the curvature hints. The curvature hints drawn by the user as 2D curves are expressed per-pixel by expanding them over the selected region (further details in Section 4.4). The advantage of this approach is that depth maps are intrinsically free of topological and geometrical inconsistencies (e.g. self-intersections), since the energy value and its gradient only depend on the depth values. In addition, the existing 3D reconstruction together with the curvature constraints reduce the ballooning effects typical of many energy-based methods. Finally, the GPU implementation guarantees a fast computation of the final

surface.

4.3 Segmentation on Calibrated Images

Our approach is mostly a direct extension of Rother et el. [98] to the case of multi-view datasets. Unlike previous approaches, 3D points are not only a way to connect pixel of different images, but are also elements that are classified. Therefore the user input is a selection of points or pixels, depending on which operation is easier on the given dataset. Let V be the set of reconstructed vertices. We know that each vertex in V corresponds to a pixel in two or more images, that is, the pixels that were matched to infer the 3D position of the vertex. So we indicate with $Corr(v)$ the set of pixels corresponding to vertex v. Let $G = (V \cup P, E)$ be a undirected graph where V is the set of input vertices and P is the set of pixels of all images. The set of edges E is defined as:

$$\begin{aligned} E = &\{(p_i, p_j) | p_i \; adjacent \; to \; p_j\} \cup \\ &\{(p_i, v_j) | p_i \in Corr(v_j), v_j\} \cup \\ &\{(v_i, v_j) | \|v_i - v_j\| < \tau\} \end{aligned} \tag{4.1}$$

where we use p to refer to pixels and v for vertices. The first type of edge is the regular pixel-pixel edge used in the single image segmentation algorithm. The second type connects pixels that were matched to create a vertex with the vertex itself, thereby generating a 2D-3D connection. Finally, the third edge type connects vertices that are closer than a threshold in 3D space. This means that a selection made in one image can *propagate* through space and end up in other images. Choosing the weights for the pixel-vertex and the vertex-vertex edges entails taking into account the specific algorithm. Here, we use this general idea to extend the GrubCut [98] algorithm.

The GrabCut algorithm [98] employs two Gaussian Mixture Models (GMM), one for the foreground and one for the background, to model color distribution. These GMMs are mixtures of K full-covariance Gaussians. A vector $\alpha \in \{0, 1\}^N$ assigns to each element (pixel) a flag indicating foreground or background. In addition, Rother et al. [98] introduced the idea of using also a vector $\underline{\mathrm{k}}$ assigning an element to a specific component of the mixture model. The element themselves, that in our system can either be pixels or vertices, are indicated with vector $\underline{\mathrm{z}} \in P \cup V$. The algorithm proceeds by globally minimizing a Gibbs energy function of the form

$$E_s(\alpha, \underline{\mathrm{k}}, \theta, \underline{\mathrm{z}}) = U(\alpha, \underline{\mathrm{k}}, \theta, \underline{\mathrm{z}}) + V(\alpha, \underline{\mathrm{z}}) \tag{4.2}$$

where θ are the parameters of the Gaussian mixtures, i.e. the weights for the components, and the means and covariance matrices of the individual components. The data term U is computed as in the original formulation by evaluating the log-likelihood w.r.t the assigned Gaussian from the GMM, with the difference that our elements can be vertices in addition to pixels. More precisely:

$$U(\alpha, \underline{\mathrm{k}}, \theta, \underline{\mathrm{z}}) = \sum_N (-\log p(\underline{\mathrm{z}}_n | \alpha_n, \underline{\mathrm{k}}_n, \theta) - \log \pi(\alpha_n, \underline{\mathrm{k}}_n)) \tag{4.3}$$

The smoothness term is instead specified with respect to an augmented neighborhood system C, which takes into account 2D-2D links between adjacent pixels (the

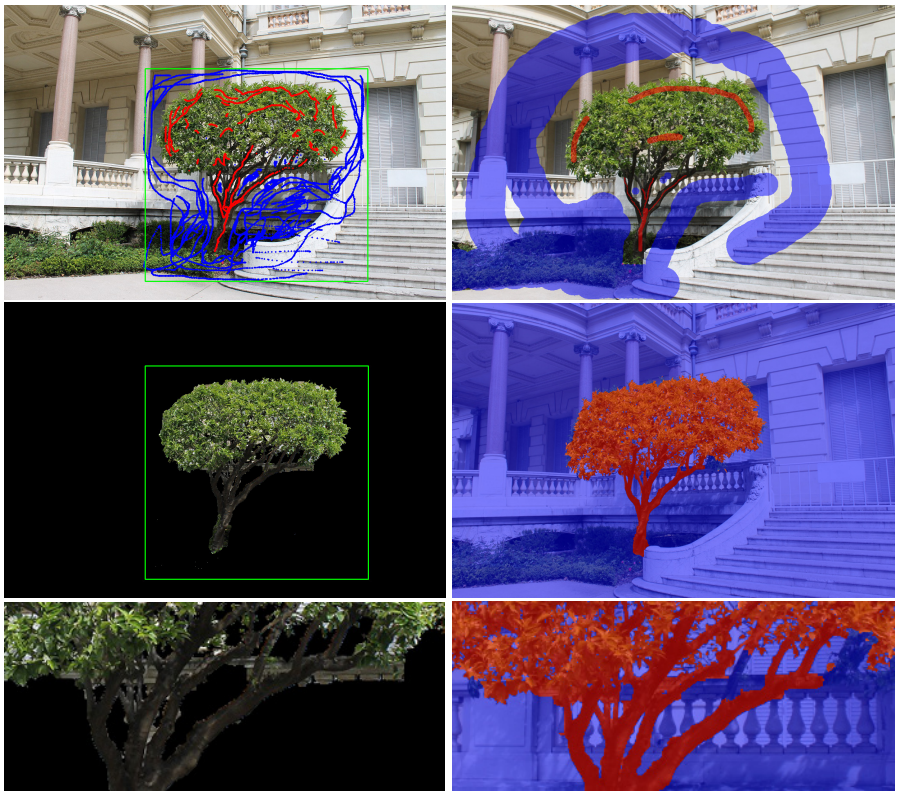


Figure 4.1: *The proposed algorithm vs the standard Grabcut algorithm (1st row - user input, 2nd row - result, 3rd row details). Note that only 1 out of 27 images received input from the user.*

only type of link in the GrabCut), 3D-2D links between a vertex and its projection onto the images and 3D-3D links, between neighbor vertices in 3D space.

$$V(\alpha, z) = \gamma \sum_{i,j \in C} \begin{cases} e^{-\beta||z_i - z_j||} & \text{if } i \in P, j \in P \\ \Gamma & \text{if } i \in P, j \in V \\ e^{-(\beta||z_i - z_j|| \tau ||p_i - p_j||)} & \text{if } i \in V, j \in V \end{cases} \tag{4.4}$$

Constants β is set as in the 2D case, that is:

$$\beta = 2 \ (\llbracket z_i - z_j \rrbracket)^{-1} \tag{4.5}$$

where $\llbracket . \rrbracket$ indicates the expected value, and z is in CIElab color space, and constant τ for the 3D-3D links is found by extending the same idea as $\tau = 2 \ (\llbracket p_i - p_j \rrbracket)^{-1}$ where p is the position in 3D space. Γ is a big constant that ensures that a pixel and its projected vertex do not get classified in different sets.

The background and foreground pixels marked by the user are used to initialize two Gaussian Mixture Models for the foreground color distribution and the background color distribution. This initialization is performed with a k-means algorithm. The algorithm then iteratively performs the following operations:

1. assign each unknown pixel and vertex either to the background or to the foreground GMM (estimate α)

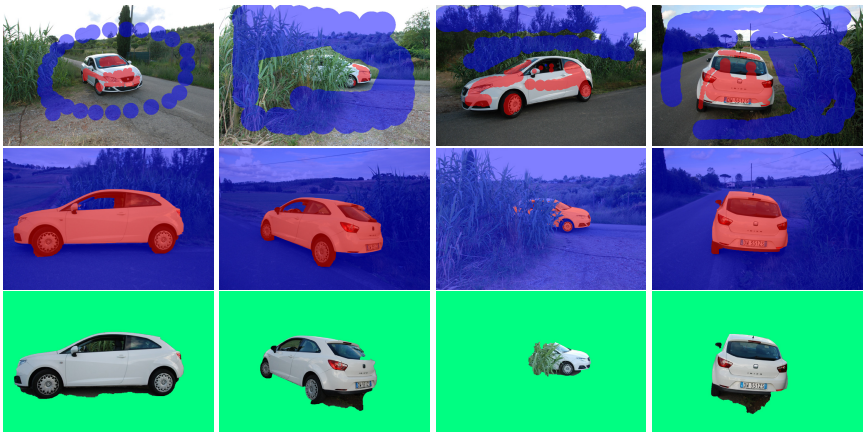


Figure 4.2: *Example of joint 2D-3D dataset segmentation. The top row shows the 4 images out of 55 where the user provided input for segmentation. The second row shows some of the output image and the third row the same image with the technique introduced in [32].*

2. assign a specific Gaussian within the assigned GMM to each pixel/vertex (estimate $\underline{k}$)
3. re-estimate mean and covariance for each Gaussian in the GMM based on assigned pixels/vertices (estimate θ)
4. solve minimization by GraphCut, estimating sink and source energies according to the GMMs (estimate α)
5. repeat from step 2 until convergence

Figure 4.1 shows some results of our technique vs the Grab Cut approach. As expected, the multi-view version is more accurate and requires less precise user interaction. This is the obvious consequence of using multiple images in our setting. Additionally, we show in Figure 4.2 a comparison with an automatic state of the art multi-view image segmentation method, i.e. the approach presented in [32], on a dataset where the object of interest, the car, is not fully visible in all of the images. It can be seen how, although both the algorithm produce acceptable results, our approach is able to correctly classified the car's pixel even behind vegetation. Moreover, the technique [32] applied to the Museum dataset (the one used in Figure 4.1) cannot produce any usable result due to the fact that the object of interest (the tree) is not in the center of the images.

4.4 Defining Curvature Hint

The aim of the curvature hints provided by the user is to know, for a specific subregion of the image, that the directional curvature along certain directions is zero.

In order to better explain how these curvature hints work we will refer to a practical example. Figure 4.3 shows a drawing of a gas pipe. The user wants to hint that the surface curvature is zero along the direction of the pipe and does so by drawing a line like the red one shown in the figure. The intended meaning is that on any location of the line the curvature along the tangent vector is zero. The other points of the region inherit the direction vector of their closest point on the line so as to obtain a vector field

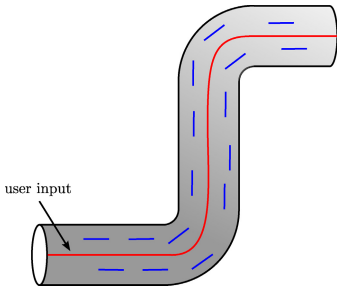


Figure 4.3: *Scheme of the curvature hint. The user input is shown as a red line, the inferred vector field with blue short segments.*

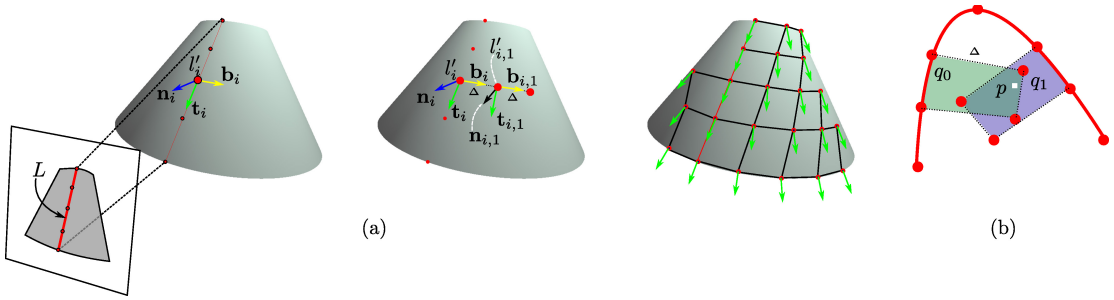


Figure 4.4: *Curvature hint propagation. (a) From a user-given line in image space to a regular sampling on the surface with inferred directions. (b) Inferring the directions in image space. Point p is included both in q_0 and in q_1, but it is closer to segment s_1 and thus the corresponding direction value is used.*

(described by blue segments in the figure). In other words, we infer a vector field for all the pixels of the region of interest starting from the input curvature constraints.

To achieve this, first, the input line L is sampled as $L_s = \{(l_0, t_0), \ldots, (l_k, t_k)\}$, where l_i is the position of the sample and t_i is the tangent vector at l_i. Then the samples are projected onto the 3D surface obtaining $L'_s = \{(l'_0, n_0, t'_0), \ldots, (l_k, n_i, t_k)\}$ where l'_i is the projection of l_i, n_i is the surface normal at l'_i and t'_i is the projection of t_i on the tangent plane passing through l'_i, that is, the tangent plane at l'_i. So far, we have obtained the direction of the zero curvature for the sampled points. The next step is to propagate this information to the rest of the surface. We obtain the direction orthogonal to t'_i as $b_i = n_i \times t_i$ and then define a new grid node $(l'_{i,1}, n_{i,1}, t'_{i_1})$ where: $l'_{i,1}$ is the 3D point found by walking a distance Δ from $l'_{i,1}$ along b_i, $n_{i,1}$ is the normal at $l'_{i,1}$ and $t'_{i,1} = b_i \times n'_{i,1}$. We iterate this process along b_i and $-b_i$ for all i until we obtain a grid covering the projection of the selected region on the surface, where the direction of zero curvature and the distance from the point on the same line of the grid is associated with each node.

The final step is to interpolate the directions stored at the grid nodes for all the pixels. This can be simply done using rasterization, by rendering the grid as filled quads and letting attribute interpolation do the work. However, note that, unless the line is a straight one, pixels would be covered by more than one quads, while our goal is to get the value from the closest point on L'_s (see Figure 4.4). This problem is solved using an idea proposed by Hoff et al. [56] to compute a voronoi diagram of points and lines. To each grid node we assign a z coordinate (in view space) as its distance from the line L'_s. From the fragments with the same coordinates, the depth test will thus automatically

return the one closest to the line.

4.5 Reconstruction

The reconstruction algorithm takes as input the reconstructed geometry, the selection as defined in Section 4.3, and the curvature hints. Then, it gives in output a set of depth maps $Z = \{\mathbf{z}_h | h = 0 \dots N\}$, that is the depth values for the pixels of the regions r_h, which altogether form the final reconstructed surface.

The algorithm proceeds by minimizing a four term energy function $E(Z)$, defined as

$$E(Z) = \sum_{\forall h} S(\mathbf{z}_h) + F(\mathbf{z}_h) + R(\mathbf{z}_h) + C(\mathbf{z}_h) \tag{4.6}$$

$S(\mathbf{z}_h)$ is a term to ensure the smoothness of the surface, $F(\mathbf{z}_h)$ is a term to ensure that the surface approximates the original points, $R(\mathbf{z}_h)$ ensures that different depth maps agree on overlapping regions and $C(\mathbf{z}_h)$ accounts for the curvature hints given by the user.
The minimization is carried by iterative gradient descent. Since we want to leverage on the graphics hardware, we perform the gradient descent computation camera by camera, that is:

$$\begin{aligned} \mathbf{z}^{i+1}_{|h} &= \mathbf{z}^i + \alpha_{i,h} \nabla E(\mathbf{z}^i_{|h}),\ h = 0 \dots k \\ \nabla E(\mathbf{z}^i_{|h}) &= \nabla S(\mathbf{z}^i_{|h}) + \nabla F(\mathbf{z}^i_{|h}) + \nabla R(\mathbf{z}^i_{|h}) + \nabla C(\mathbf{z}^i_{|h}) \end{aligned} \tag{4.7}$$

We use an adaptive step size $\alpha_{i,h}$ which varies with the global energy as proposed in [12] and increases the convergence rate:

$$\begin{aligned} \alpha_{i,h} &= \frac{(\mathbf{z}^i_{|h} - \mathbf{z}^{i-1}_{|h}) \cdot \Delta^i}{\Delta^i \cdot \Delta^i} \\ \Delta^i &= \nabla E(\mathbf{z}^i_{|h}) - \nabla E(\mathbf{z}^{i-1}_{|h}) \end{aligned} \tag{4.8}$$

Note that the unknowns are the depth values of all the pixels in the ROI. We formulate these energy terms in such a way that the gradient $\nabla E(\mathbf{z}^i)$ can be evaluated in the GPU for each depth map. In the following we define each of these terms, while we address the interested reader to the Appendix A for the complete algebraic derivations of the gradient term.

Please note that the formulas for energies involve derivations of the depth maps $\mathbf{z}_h$ along x and y in image space, more specifically second order derivatives $\frac{\partial \mathbf{z}_{xx}(i,j)}{\partial z_{m,n}}$ $\frac{\partial \mathbf{z}_{xy}(i,j)}{\partial z_{m,n}}$ and $\frac{\partial \mathbf{z}_{yy}(i,j)}{\partial z_{m,n}}$, while we will need the gradient of these energies with respect to the z value, that is, the depth of each pixels, which are the unknown variables of the system.

4.5.1 Smoothness Term: $S(\mathbf{z}_h)$

The smoothness term is defined as the *thin plate* energy:

$$\begin{aligned} S(\mathbf{z}_h) = &\sum_{i,j} \mathbf{z}^2_{xx}(i,j) + \\ &2\mathbf{z}^2_{xy}(i,j) + \mathbf{z}^2_{yy}(i,j) \Delta x \Delta y \end{aligned} \tag{4.9}$$

where we dropped the pedix h to simplify the notation.

4.5.2 Approximation Term: $F(\mathbf{z}_h)$

The aim of the approximation is to make the final surface an approximation of the initially reconstructed one. We use the implicit Moving Least Squares formulation proposed in [93] to define the surface approximating the input points. With MLS, the surface is the zero set of a function F, $s = \{x|F(x) = 0\}$. At initialization time, we sample the value of ∇F in the neighborhood of the input points, storing the result in an octree. Note that this is the only term for which we can pre-compute and store the gradient because it does not depend on the depth values but only on the input points.

4.5.3 Coherence term: $R(\mathbf{z}_h)$

The coherence term, forces two depth maps to coincide on their overlapping 3D region. In general, depth maps corresponding to the same portion of the real surface do not actually match. This is due to two factors: first, each depth map is obtained by interpolating a different set of vertices, although all of them are approximately on the real 3D surface (see also Section 4.5.5); second, even if the set of vertices were the same, the image space discretization of the depth maps would induce large aliasing effects, especially near the silhouette of each region where the surface is steep w.r.t.the camera point of view.

The coherence term is defined as:

$$R(\mathbf{z}_h) = \sum_{\forall k} \sum_{ij} \left[\phi\left(g_k(i,j,\mathbf{z}_{i,j}) - f_k(i,j,\mathbf{z}_{i,j})\right)\right]^2 \tag{4.10}$$

where $g_k(i,j,\mathbf{z}_{ij})$ gives the depth of the projection on the neighbor camera c_k of the un-projection of the triple $i,j,\mathbf{z}_{ij}$ of the camera h and $f_k(i,j,\mathbf{z}_{i,j})$ is the current depth value stored at the same location, as illustrated in Figure 4.5. ϕ is a threshold function used to define when two depth values are close enough to enforce them to be the same. Note that the outer summation of equation 4.10 runs over the cameras that overlap with

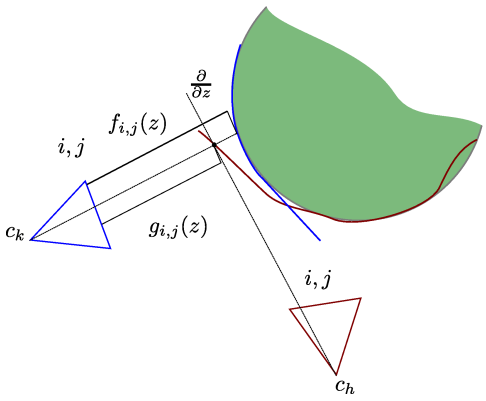


Figure 4.5: *Enforcing coherence between overlapping range maps.*

h. Typically, for each pixel in r_h, there are from 0 to 5 neighbors.

4.5.4 Curvature term: $C(\mathbf{z}_h)$

The specified directions of zero curvatures are indicated with $\mathbf{u} = [u, v]$ for the generic pixel and are obtained as explained in Section 4.5.4. In order to simplify the formula and avoid floating point divisions, instead of minimizing the actual curvature we use the square of the directional second order derivative of $\mathbf{z}_h$, which is:

$$C(\mathbf{z_h}) = \sum_{ij} (\mathbf{u}^T H(\mathbf{z_h}_{i,j})\mathbf{u})^2 \tag{4.11}$$

where H is the *Hessian* matrix of z_h.

Note that the direction vector has to be specified for all the cameras. However, it would be very tedious for the user to manually define the directions of zero curvature for each image and, worse, it is unlikely to be consistent for all the cameras. Therefore we propagate the vector $\mathbf{u}_h$ on all the other cameras by projecting it in world space and hence re-projecting it on each camera.

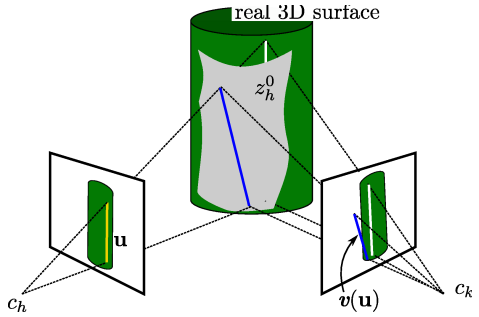


Figure 4.6: *Propagation of the user-given hint on the direction of zero curvature.*

This is achieved using the depth map z_h obtained by the initialization phase and therefore the approximation of the depth map will influence the projection (we recall that only the smoothing term is considered at the initialization phase). This means that the original vector and its propagation on the other cameras could be inconsistent on the real 3D surface, as shown in Figure 4.6. To resolve this inconsistency, we perform the propagation of the vector at each iteration, so that it tends to converge along with the convergence of all the depth maps.

The full formula is written as:

$$C(\mathbf{z_h}) = \sum_{ij} \left(z_{xx}(i,j)u^2 + 2\, z_{xy}(i,j)uv + z_{yy}(i,j)u^2 \right)^2 \tag{4.12}$$

4.5.5 Initialization

Since we are using gradient descent, it is crucial to find an initial solution, i.e. an initial depth map for each camera, which is close to the global minimum and at the same time approximates the input reconstructed points. If the projection of the input points on a given camera is dense (as can happen, for example, using the output of PMVS, see the Toy Car example in Figure 4.9), we can simply triangulate the projections and obtain the depth map. On the other hand, if the starting point is a sparse reconstruction (as with the Pipe and the other models in Figure 4.9) we minimize the energy term $S(\mathbf{z}_h)$

by imposing the input points as hard constraints, which is done by solving the resulting system $\nabla S(\mathbf{z}_h)$ as shown in [118].

4.5.6 Handling Discontinuities in the Depth Maps

Note that using finite differences for approximating 2^d order derivatives always gives an expression of the following form:

$$\nabla(E(Z))_{ij} = \sum_{h,k=-2}^{i,j<2} w(h,k)z(i+h,j+k). \tag{4.13}$$

This means that the gradient at pixel (i,j) is a linear combination of values in a 5×5 kernel centered at (i,j). In the derivation of the energy terms shown so far, we have always employed central differences to express the partial second derivatives contained in the formula, i.e. $\frac{\partial E(z)}{\partial xx}, \frac{\partial E(z)}{\partial yy}, \frac{\partial E(z)}{\partial xy}$. Unfortunately, depth maps have borders, so we have to adjust the computation of second derivatives on border pixels. A pixel may be on a border in two cases: either because the adjacent pixels are not part of the ROI or because there are discontinuities in the depth maps. The latter case is detected by a check that is run at the beginning of each minimization step, by testing if the difference with one of their adjacent pixels is above a certain threshold. Typically, these discontinuities appear as the surface evolves, while at the beginning, triangulation alone or the thin plate energy tend not to create very steep surfaces.

In order to handle discontinuities, we write the energy term $E(z_h)$ so that differential quantities are discretized using central, forward or backward differences adaptively, depending on which adjacent pixels are available.

This is done by computing a bit code for each pixel, that is, a *tag*, indicating how each differential quantity must be computed. Listing 4.1 shows the algorithm , CxCx, FxFx and BxBx stand for central, forward and backward difference and NOxx means that it is not possible to compute the second order derivative on that pixel.

Listing 4.1: *Algorithm to create the code for the 2nd order derivative along x.*

```
CodeDxDx(i,j)
{
  if( (i+1,j) in r_h && (i-1,j) in r_h )
    tag = CxCx;
  else if( (i+2,j) in r_h && (i+1,j) in r_h )
    tag = FxFx;
  else if( (i-1,j) in r_h && (i-2,j) in r_h )
    tag = BxBx;
  else
    tag = NOxx;
}
```

This means that the exact expression of $\nabla(E(Z))_{nm}$ depends on how the differential quantities in the neighbor pixels are computed. For each pixel of coordinates (i,j), we compute a code that says how $\mathbf{z}_{xx}$, $\mathbf{z}_{yy}$ and $\mathbf{z}_{xy}$ are computed. In a static LUT, for each code, we store the coefficients to be applied to the neighborhood pixels. Figure 4.7 shows an example of three entries of the LUT (see Appendix B for the derivation of these coefficients).

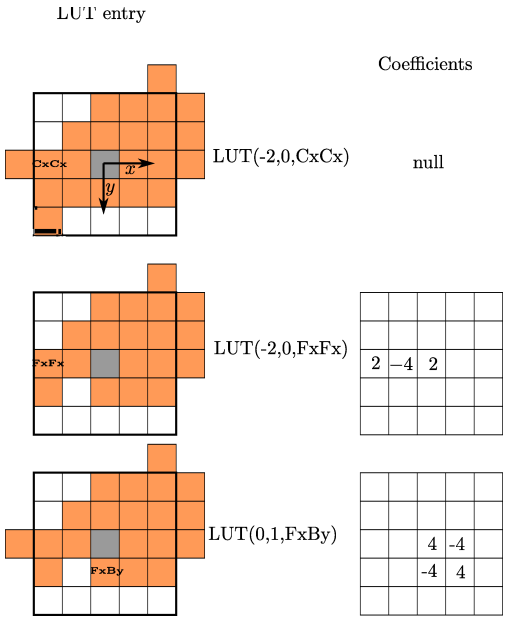


Figure 4.7: *Example LUT entries for computing the gradient of the smoothness term on a pixel (the central one).*

4.5.7 Performing the Iterative Minimization

Figure 4.8 shows a scheme of the iterative minimization algorithm for a generic camera. We use a fullscreen quad to enable the fragment shader to output a value for each single $z_{n,m}$. Each iteration of the minimization is performed in four steps. In the first pass the tag values explained in Section 4.5.6 are computed and stored in the alpha channel of the target buffer `BufferTag`.

In the second pass, `BufferTag`, which contains the current and previous solution and the previous gradient, is bound as texture. The fragment shader `FS_ComputeDelta` computes the gradient $\nabla E(\mathbf{z}^i)$, passes along the value $\mathbf{z}^i$ and computes a *first part* of the Equation 4.7 (that is, the components to be summed to obtain the dot product).

The third pass consists of summing all the values of the componentwise products to obtain the two factors of the fraction in Equation 4.7. This is done by building a texture pyramid, as with mipmapping but summing the four texel values instead of averaging them. Then the final 1×1 texture is readback in the main memory and α_i can be computed. The fourth and last step consists of bounding the `BufferStepSize` as texture and `BufferPos` as target and computing $\mathbf{z}^{i+1}$ and copying $\mathbf{z}^i$ and $\nabla E(\mathbf{z}^i)$.

Note that when computing $\mathbf{z}^{i+1}_{|h}$ we need to keep in video memory only the depth map h and the maps overlapping with h, which are normally less than five. The need for the overlapping depth map is due to the only term that is not separable over the depth maps, that is, the coherence term $R(\mathbf{z})$.

To speed up convergence rate we also use a V-Cycle multigrid method (see [16]) on each camera. The multigrid approach consists of transferring the solution found for the original depth (grid) into a coarser grid (*restriction phase*), performing some minimization steps and then transferring back the solution to the finer grid (*interpolation phase*).

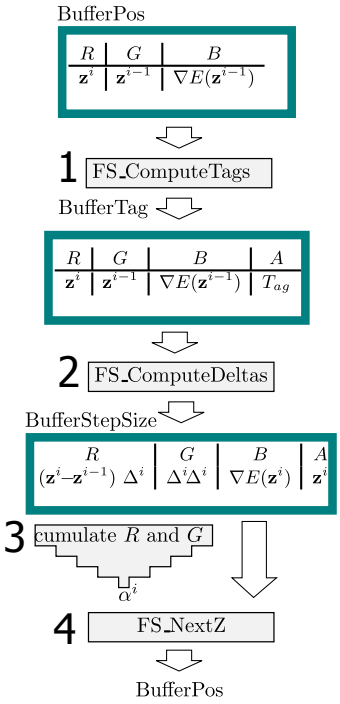


Figure 4.8: *The flow chart of the minimization algorithm.*

4.6 Results

We tested our system on several datasets, acquired from urban scenarios or daily life objects. Here, we highlight five examples: a pipe, a garbage bin, a van, a toy car and a plastic bath seat (for babies). Figure 4.9 shows on the first column a sample image of the object, on the second the input used by our system. In these experiments we used both dense and sparse reconstruction to demonstrate that the approach can work well in both cases. The input for the Pipe, the Garbage Bin, the Van are the points reconstructed by Bundler [110] after the camera calibration (note that these points are quite dense for the Van). Instead, for the Toy Car and the Bath Seat we use the output of the PMVS algorithm as input. For comparison purposes, the images of the third row are produced by the Poisson reconstruction algorithm [69] run on the 3D points reconstructed using the PMVS [42]. Please note that the results obtained with the PMVS+Poisson reconstruction are of a poor quality w.r.t our final reconstruction (shown on the fourth row) also when the shape is quite complex like for the Bath Seat case. For the Pipe, the Garbage Bin and the Bath Seat we needed only one constraint to specify the direction of zero curvature (see Table 4.1). Note also that, even if the input points are few and they are irregularly distributed (especially in the Pipe test), they are sufficient for a good initialization of our reconstruction algorithm.

Figure 4.10 shows two images of the reconstruction of the Pipe dataset. It can be easily seen how not imposing any constraint (left image) results in a deformed model where, although the smooth and overlapping energy terms are minimized, the final shape does not correspond to the one of the original pipe.

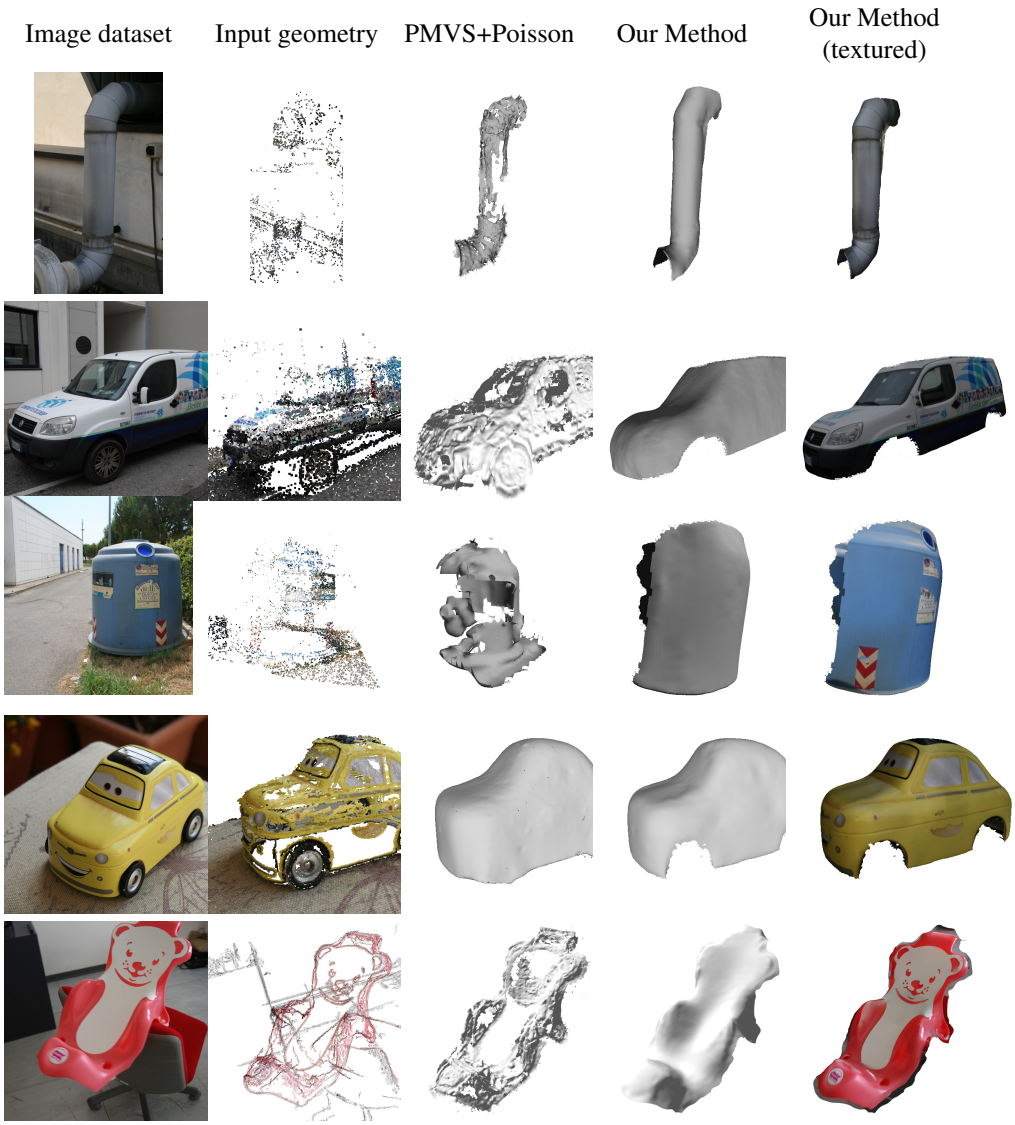


Figure 4.9: *Results of our experiments (last two columns) and comparison with reconstructions obtained by applying the Poisson surface reconstruction algorithm [69] to the output of the PMVS algorithm [42] (3rd column).*

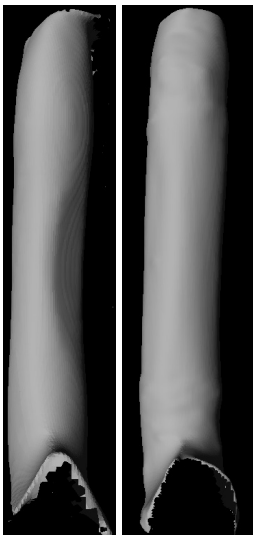


Figure 4.10: *Reconstruction of the Pipe without any curvature hint (Left) and with curvature hint (Right). Note that the reconstruction without constraints does not meet the cylindrical shape of the original pipe due to small reconstruction errors in the input points.*

4.6.1 Computation time

The lifecycle of a reconstruction consists of the following steps

1. The user selects the region of interest on one or more images.
2. The segmentation algorithm described in Section 4.3 and the initialization of range maps are run.
3. The user provides curvature hints as described in Section 4.4.
4. The reconstruction phase is run.

where steps 1-2 and 3-4 can be iterated to improve the final result.

Table 4.1 reports the time for the experiments run on a PC with Intel I7 4820k, 3.70 Ghz, equipped with 32 GB Ram, graphics board nvidia GeForce GTX 780 and the number of strokes provided by the user. The segmentation strokes are provided in the following way: $n_1; n_2$, where n_1 is the number of images annotated and n_2 is the total number of strokes on these images. The column "curvature hint strokes" reports the number of images annotated with the curvature constraints in the same way. For example, for the Garbage Bin we put a vertical curvature constraint on its cylindrical part on two images, for the Pipe the curvature constraint follows the pipe profile in three images. The Bath Seat requires only one curvature constraint in the center of its white lower part. The Van requires two strokes, one on the mirror and one on the hood to convey the right curvature of these two parts. It can be seen that all the times (segmentation, initialization and minimization) are roughly proportional to the number of photographs.

Note that the reconstruction time is fast (just a few seconds), while the segmentation and initialization times are quite slow. This is simply due to the fact that our system is still a prototype written in #F sharp. This limits the overall performance but not the

Table 4.1: *Model reconstruction performance (time is in seconds).*

Model	Camera (#)	Segm.	Segm. Strokes	Curvature Hint Strokes	Init.	Recons.
Pipe	34	29.56	3 ; 12	3 ; 3	42.37	16.20
Van	50	65.95	8 ; 38	2 ; 4	184.13	27.63
Garbage Bin	19	16.92	3 ; 10	2 ; 2	35.04	9.45
Toy Car	75	75.63	10 ; 40	0 ; 0	1.1	141.32
Bath Seat	28	33.205	5 ; 14	2 ; 2	829.37	59.76

reconstruction algorithm which is written entirely in GPU. The time for initialization is all due to the solution of the thin plate equation for each camera, except in the case of the Toy Car, where the initial depth map was obtained by triangulation in considerably less time. We point out that in the cases like the Bath seat, where the points density is moderate-high both the approaches can be used. Hence, in this case the initialization time can be reduced by using the triangulation approach. Also, the number of strokes is very low in all cases, thanks to the technique shown in Section 4.4 which allows us to propagate the stroke done in one photo to the neighbor cameras.

CHAPTER 5

Conclusion and Future Work

The use of 3D content is now pervasive in several engineering, marketing and communication applications. However, the presentation of 3D content is a challenging task because of the inherent complexity of this type of data. Several approaches have been proposed in literature to create convincing presentations of both scenes available in digitized form and scenes acquired using image-based methods.

In this dissertation, we tackled the problem of presenting 3D content in an intuitive and effective way from multiple angles.

5.1 Presentating 3D content using videos

In Chapter 2, we studied the case of complete 3D scenes, such as human-made CAD models, and unassisted ways to present it to the user. We have proposed an algorithm that produces a video of a static scene using another video as reference. We built on the intuition that two videos are especially perceived as similar on the base of the optical flow they produce. Therefore, we worked on a method for finding camera path in the 3D scene whose produced flow is most similar to the one generated by the example video. The processing steps of our pipeline have been finely tuned and a user study corroborates our results and the idea of using flow similarity as a criterion for video similarity.

5.1.1 Improving the presentation of 3D scenes using videos

Although we demonstrated the effectiveness of our algorithm, the final pipeline could be enhanced. Taking inspiration from the limitations underlined in Section 2.8.3 we can foresee several paths to improvement.

Views selection using saliency metrics and user annotations

We could extend our approach in a way that it may guarantee that specific areas of interest of a 3D scene appear in the final video. This can be obtained by constraining the search space to instant flows contained in particular areas. Such areas could be selected in two ways:

- Using a view saliency selection algorithm such as [104].
- By providing an intuitive interface to let the user annotate interest points on the 3D model.

Of course, a combination of the two approaches is possible.

Generalization of the camera path generation

Our algorithm produces a camera path starting from the camera trajectory recovered by a camera tracker and then finding a similarity transformation that once applied to the original trajectory minimizes the difference between the optical flow of the example video and the motion flow calculated from the 3D scene. In this scenario there are two limitations:

- Camera tracking is still problematic for challenging video sequences and the resulting camera path can be very noisy or inconsistent.
- The parametrization of our algorithm is restricted to a similarity transformation. More general transformation with higher degrees of freedom could be implied to avoid being stuck in local minima.

To solve the above problems, a direction of research would consist in removing the camera tracking from the pipeline as a whole and not generating the paths as a transformation of input estimated camera path. Instead, we could try an incremental construction of the path starting by connecting nearby instant flows and extending the path as a function of the flow difference with the video.

Multiple shots generation and montage

Our algorithm could be easily extended to take in input a video composed of multiple shots. In this case, we should implement one of the many state-of-the-art video shots segmentation algorithms (see [107] for a review). Once a video is segmented in shots, our pipeline can be applied unchanged to each shot. Then, all the output shots can be reassembled to form the final sequence. Given the functioning of our system, all output shots and the reassembled sequence have the same durations of the input. Thus also the montage is implicitly cloned. Despite this process can be effective, we foresee the following problems:

- All shots contained in the input video must be suitable for the desired presentation; otherwise the user should manually segment and remove the unwanted shots.
- It can be tedious for a user to find a video sequence which has both the right "style" and duration.

We envision a better way to produce videos composed of multiple shots. A very practical approach should take in input the following items: a set of shots taken from one or multiple videos, the desired duration in seconds and the 3D scene of interest. For each shot, our original algorithm is applied unchanged. The produced shots are then fed into an algorithm which performs their montage to generate a sequence with the desired user duration. This novel "montage module" should not only perform cuts where required but also choose an appropriate ordering of the shots and how to transition from one shot to the other.

Lighting transfer from video

Filmmaking is a complex task where different factors contribute to the final "look and feel" of the video. The method presented in this thesis unburden the user from controlling the virtual camera. However, other elements come into play when shooting a scene. In particular, the setup of light sources plays a central role. As lighting design is a complex and tedious task even for the more experienced users, semi-automatic or fully automatic solutions soon became subjects of great interest. Several approaches already exist in literature (see [102] for a survey) and they can be classified into two classes: inverse lighting and lighting by example. Inverse lighting starts from a user specification of the lighting requirements usually by using painting interface [94, 95]. By using various optimization strategies [38, 39, 47], these systems try to solve the "Light Source Emittance Problem" (EP) or the "Light Source Positioning Problem" (LSP) or both problems at same time [47]. The lighting by example approach is instead more related to our work as it tries to transfer or mimic the lighting from a reference image [52]. To the best of our knowledge, no approaches have been proposed that transfer constant or varying illumination from a reference video to a video presenting a 3D scene. To accomplish this task, an algorithm should take into account not only the positions, orientations and intensities of lights, but also constraints relative to the movement and lens parameters (for example the exposure) of the cameras.

5.2 Presenting 3D content from images: stereoscopic navigation

In Chapter 3, we presented a novel method for presenting 3D scenes acquired from the real world though image-based methods. Our first contribution in this field is the key insight of exploiting the spatial organization of a set of registered cameras to build the *StereoSpace* formalization and its extension to a continuous domain via the use of an IBR engine. Thanks to this framework, we developed a system to navigate a photo collection using stereoscopy as the means to provide an immersive experience without the need of recovering the actual 3D surfaces of the depicted scene.

5.2.1 Improving the presentation of 3D scenes from images

Several improvements are also possible for the methods presented in Chapter 3 and Chapter 4, we present here the most promising.

3D Scenes navigation with omnidirectional images

The idea of using 360° panoramas for virtual reality applications is well known [24]. However, recent improvements in the quality and affordability of both VR displays

and panoramic capture devices renewed the interest in panoramic images and videos. Moreover, novel IBR techniques have been proposed to generate monoscopic or stereoscopic images of a scene acquired with a set of omnidirectional panoramas arranged in a lattice structure or moving along a path [6, 58, 124]. It would be interesting to extend the *StereoSpace* formulation to a five-dimensional space in which the user is not only able to zoom and pan the camera but also to rotate it with respect to up direction and to tilt it.

5.3 Presenting 3D contents from images: geometry reconstruction

Our final contribution, in Chapter 4 is a user-assisted multi-view 3D reconstruction framework. Here, the key idea is to integrate user hints into our novel energy-based reconstruction engine. User hints are used for two purposes: to perform a foreground/background joint 2D-3D segmentation of the input dataset and to specify the nature of surfaces being reconstructed. In particular, we devised a method to integrate into our algorithm hints on the directions of zero curvature of the surfaces to reconstruct. We demonstrated that our method is particularly useful in the reconstruction of human-made artifacts and can turn a poor-quality reconstruction produced with state of the art image-based reconstruction techniques into a high-quality one.

Multi-hint reconstruction framework and inverse CAD

A natural evolution of our framework would be to add more possibilities for user hints, for example, to indicate sharp features or straight segments. We also aim to modify the initialization phase of the reconstruction by integrating silhouette-based reconstruction methods to obviate the need for an initial sparse reconstruction. An exciting direction of research would be to use our work to tackle the inverse CAD problem; that is: turning the output of MVS algorithms into a 3D parametric representation to ease the human post-processing of such 3D data. As a matter of fact, as we saw in Chapter 4, MVS algorithms can produce dense and accurate 3D models. However, such 3D models are not suitable for CAD tools as they do not use a parametric representation. Although some solutions have been proposed, like [89], turning such results in accurate and manageable CAD representations is still an open issue with no practical solutions at hand [44].

Appendices

APPENDIX $\mathcal{A}$

Algebraic derivations of the energy terms gradient

In this appendix, we present the full derivations of the energy terms gradients from Section 4.5.

A.1 Smoothness term

The derivative of the smoothness term with respect to $z_{m,n}$ is simply:

$$\begin{aligned}\frac{\partial}{z_{m,n}} S(\mathbf{z}_h) = \sum_{i,j} 2\mathbf{z}_{xx}(i,j)\frac{\partial \mathbf{z}_{xx}(i,j)}{\partial z_{m,n}} + \\ 4\mathbf{z}_{xy}(i,j)\frac{\partial \mathbf{z}_{xy}(i,j)}{\partial z_{m,n}} + \\ 2\mathbf{z}_{yy}(i,j)\frac{\partial \mathbf{z}_{yy}(i,j)}{\partial z_{m,n}}\Delta x \Delta y\end{aligned} \tag{A.1}$$

The unrolled formula, using central finite differences, is:

$$\begin{aligned}\nabla E_s(\mathbf{z})_{m,n} = \frac{\partial}{\partial z_{m,n}} \sum_{i,j} \left(z_{i+1,j} - 2z_{i,j} + z_{i-1,j}\right)^2 + \\ \frac{1}{8}\left(z_{i-1,j-1} - z_{i-1,j+1} - z_{i+1,j-1} + z_{i+1,j+1}\right)^2 + \\ \left(z_{i+1,j} - 2z_{i,j} + z_{i-1,j}\right)^2\end{aligned} \tag{A.2}$$

which finally gives:

$$\begin{aligned}\nabla E_s(\mathbf{z})_{m,n} = 25\mathbf{z}_{m,n}+ \\ \frac{3}{2}\left(\mathbf{z}_{m,n+2}+\mathbf{z}_{m,n-2}+\mathbf{z}_{m+2,n}+\mathbf{z}_{m-2,n}\right)+ \\ -8\left(\mathbf{z}_{m+1,n}+\mathbf{z}_{m-1,n}+\mathbf{z}_{m,n+1}+\mathbf{z}_{m,n-1}\right)+ \\ \frac{1}{4}\left(\mathbf{z}_{m+2,n+2}+\mathbf{z}_{m+2,n-2}+\mathbf{z}_{m-2,n-2}+\mathbf{z}_{m-2,n+2}\right)\end{aligned} \tag{A.3}$$

A.2 Coherence term

For the coherence term we have:

$$\begin{aligned}\frac{d}{d\mathbf{z}_{i,j}}R(\mathbf{z}_h) = \frac{d}{d\mathbf{z}_{i,j}}\left(\left(g_k(i,j,\mathbf{z}_{i,j})-f_k(i,j,\mathbf{z}_{i,j})\right)^2\right) = \\ 2(g_k(i,j,z)-f_k(i,j,z))(\frac{d}{dz}g_k(i,j,z)-\frac{d}{dz}f_k(i,j,z))\end{aligned} \tag{A.4}$$

So we need the derivatives of $g_k(i,j,z)$ and $f_k(i,j,z)$.

Since the cameras are calibrated we know the matrix $\mathbf{R}_{h,k}$ that transform the depth values from camera k to camera h:

$$\mathbf{R}_{h,k} = \mathbf{I}_k\mathbf{E}_k\mathbf{I}_h^{-1}\mathbf{E}_h^{-1} \tag{A.5}$$

where $\mathbf{I}$ and $\mathbf{E}$ are the intrinsic and extrinsic matrices of camera h and k.
$g_k(i,j,z)$ is defined as:

$$g_k(i,j,z) = s_w\cdot\mathbf{v}(z,i,j) = \left(\mathbf{r}_{30}i+\mathbf{r}_{31}j+\mathbf{r}_{33}\right)z+\mathbf{r}_{32} \tag{A.6}$$

where $\mathbf{s}_w$ the row vector that selects component w, (i.e. $\mathbf{s}_w = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}$) and the $\mathbf{r}_{ij}$ are the components of the $\mathbf{R}$ matrix. The derivative of g_k is then:

$$\frac{d}{dz}g_{i,j}(z) = \left(\mathbf{r}_{30}i+\mathbf{r}_{31}j+\mathbf{r}_{33}\right) \tag{A.7}$$

It is no surprise that the derivative does not depend on z, because function $g_k(i,j,z)$ simply returns the distance between a point along a line and a plane, which varies linearly.

For function $f_k(i,j,z)$ things are a little harder, because it describes the depth map z_k along the projection of the line on the image plane of camera k. Let us define the parametric function describing such a projection:

$$\mathbf{u}(z):\mathbb{R}\to\mathbb{R}^2 = \frac{\mathbf{s}_{xy}\cdot\mathbf{v}(z)}{\mathbf{s}_w\cdot\mathbf{v}(z)} \tag{A.8}$$

Function f is then the composition of $z(x,y):\mathbb{R}^2\to\mathbb{R}$ with $\mathbf{u}$, i.e. $(z_k\cdot\mathbf{u}):\mathbb{R}\to\mathbb{R}$. Therefore, the derivative of the composition, is

$$\begin{aligned}\frac{d}{dz}f_k(i,j,z) =& (z_k\cdot\mathbf{u}(z))' = \left[\frac{\partial z_k}{\partial x},\frac{\partial z_k}{\partial y}\right]\cdot \\ & (\mathbf{u}_x(z),\mathbf{u}_y(z))\cdot\left[\frac{d}{dz}\mathbf{u}_x(z),\frac{d}{dz}\mathbf{u}_y(z)\right]\end{aligned} \tag{A.9}$$

We still need to define what $\frac{d}{dz}u_x(z)$ is (and, by symmetry, this will also yield its y-axis counterpart). This is the derivative

$$\frac{d}{dz}\left(\mathbf{s}_x \cdot \mathbf{v}(z)/\mathbf{s}_w \cdot \mathbf{v}(z)\right) \tag{A.10}$$

of a function with the form $\frac{\alpha z+\beta}{\gamma z+\delta}$ whose derivative is $\frac{\alpha\delta-\beta\gamma}{(\gamma z+\delta)^2}$. Therefore

$$\begin{aligned}\frac{d}{dz}f_k(i,j,z) = &\left[\frac{\partial z_k}{\partial x}, \frac{\partial z_k}{\partial y}\right]\left[\frac{\mathbf{s}_{xy}\cdot\mathbf{v}(z)}{\mathbf{s}_w\cdot\mathbf{v}(z)}\right]\cdot \\ &\left[\frac{\alpha_x\mathbf{r}_{32}-\mathbf{r}_{02}\gamma}{(\gamma z+\mathbf{r}_{32})^2}, \frac{\alpha_y\mathbf{r}_{32}-\mathbf{r}_{12}\gamma}{(\gamma z+\mathbf{r}_{32})^2}\right]\end{aligned} \tag{A.11}$$

where $\alpha_x = (\mathbf{r}_{00}i + \mathbf{r}_{01}j + \mathbf{r}_{03}) = d\mathbf{v}_x$, $\alpha_y = (\mathbf{r}_{10}i + \mathbf{r}_{11}j + \mathbf{r}_{13}) = d\mathbf{v}_y$ and $\gamma = (\mathbf{r}_{30}i + \mathbf{r}_{31}j + \mathbf{r}_{33}) = d\mathbf{v}_w$. In conclusion, the gradient is:

$$\begin{aligned}\nabla R(z)_{m,n} = &25\mathbf{z}^k_{m,n}+ \\ &\frac{3}{2}\left(\mathbf{z}^k_{m,n+2}+\mathbf{z}^k_{m,n-2}+\mathbf{z}^k_{m+2,n}+\mathbf{z}^k_{m-2,n}\right)- \\ &8\left(\mathbf{z}^k_{m+1,n}+\mathbf{z}^k_{m-1,n}+\mathbf{z}^k_{m,n+1}+\mathbf{z}^k_{m,n-1}\right)+ \\ &\frac{1}{4}\left(\mathbf{z}^k_{m+2,n+2}+\mathbf{z}^k_{m+2,n-2}+\mathbf{z}^k_{m-2,n-2}+\mathbf{z}^k_{m-2,n+2}\right)+ \\ &2(g_{m,n}(\mathbf{z}^k_{m,n})-h_{m,n}(\mathbf{z}^k_{m,n}))(\frac{d}{dz}g_{m,n}(\mathbf{z}^k_{m,n})- \\ &\frac{d}{dz}h_{m,n}(\mathbf{z}^k_{m,n}))\end{aligned} \tag{A.12}$$

A.3 Curvature term

Proceeding as for the smoothness term:

$$\begin{aligned}\nabla C(\mathbf{z})_{m,n} = &\frac{\partial}{\partial z_{m,n}}\sum_{i,j}\left(z_{i+1,j}-2z_{i,j}+z_{i-1,j}u^2+\right. \\ &\frac{2\left(z_{i-1,j-1}-z_{i-,j+1}-z_{i+1,j-1}+z_{i+1,j+1}\right)uv}{4}+ \\ &\left.z_{i+1,j}-2z_{i,j}+z_{i-1,j}v^2\right)^2\end{aligned} \tag{A.13}$$

which, after a trivial but tedious derivation, gives:

$$\begin{aligned}
\nabla C(\mathbf{z})_{m,n} = 2\,[\; & 24(u^4+v^4)+36u^2v^2)\;(z_{m,n})+ \\
& -16(u^4-u^2v^2)(z_{m+1,n}+z_{m-1,n})+ \\
& (4u^4-2u^2v^2)(z_{m+2,n}+z_{m-2,n})+ \\
& -16(v^4-u^2v^2)(z_{m+1,n}+z_{m-1,n})+ \\
& (4u^4-2u^2v^2)(z_{m+2,n}+z_{m-2,n}+ \\
& (4v^4-2u^2v^2)(z_{m,n+2}+z_{m,n-2}+ \\
& 8(u^3v+uv^3+u^2v^2)(z_{m+1,n+1}+z_{m-1,n-1})+ \\
& -8(u^3v+uv^3+u^2v^2)(z_{m-1,n+1}+z_{m+1,n-1})+ \\
& u^2v^2(z_{m+2,n+2}+z_{m-2,n-2})\;]
\end{aligned} \tag{A.14}$$

APPENDIX $\mathcal{B}$

An example of handling discontinuities with the LUT table

In this appendix, we show how the coefficients of the LUT table from Section 4.5.6 are derived in a specific case. Let us consider, Equation A.1 for the gradient of the smoothness term, which is a weighted sum of second derivatives, and consider one of the terms of the sum:

$$A = \frac{\partial}{\partial z_{m,n}} \mathbf{z}_{xx}^2(n-2,m) = 2\mathbf{z}_{xx}(n-2,m)\frac{\partial \mathbf{z}_{xx}(n-2,m)}{\partial z_{m,n}} \tag{B.1}$$

If $\mathbf{z}_{xx}(n-2,m)$ is computed by central finite differences we have:

$$\begin{aligned} &\frac{\partial \mathbf{z}_{xx}(n-2,m)}{\partial z_{m,n}} = \\ &\frac{\partial}{\partial z_{m,n}}\Big(\mathbf{z}(n-3,m) - 2\mathbf{z}(n-2,m) + \mathbf{z}(n-1,m)\Big) = 0 \\ &\Rightarrow A = 0 \end{aligned} \tag{B.2}$$

In other words, since $z_{m,n}$ does not appear in the computation of $\mathbf{z}_{xx}(n-2,m)$ the derivative on $z_{m,n}$, and thus A, is zero. Referring to Figure 4.7, this is because the entry for this configuration (first row) is null.

On the other hand, if $\mathbf{z}_{xx}(n-2,m)$ is computed by forward finite differences we have:

$$\begin{aligned} &\frac{\partial}{\partial z_{m,n}} \mathbf{z}_{xx}(n-2,m) = \\ &\frac{\partial}{\partial z_{m,n}}(\mathbf{z}(n-2,m) - 2\mathbf{z}(n-1,m) + \mathbf{z}(n,m)) = 1 \end{aligned} \tag{B.3}$$

and thus:

$$A = 2\mathbf{z}_{xx}(n-2,m) = \mathbf{2z}(n-2,m) - \mathbf{4z}(n-1,m) + \mathbf{2z}(n,m) \tag{B.4}$$

This gives as the coefficients to apply as just shown in Figure 4.7 (second row).

Bibliography

[1] Sudipta N. Sinha Adarsh Kowdle and Richard Szeliski. Multiple view object cosegmentation using appearance and stereo cues. In *European Conference on Computer Vision (ECCV 2012)*, October 2012.

[2] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Classcut for unsupervised class segmentation. *ECCV2010*, pages 8–10, 2010.

[3] C Andújar, P Vázquez, and M Fairén. Way-Finder: guided tours through complex walkthrough models. *Computer Graphics Forum*, 23(3):499–508, 2004.

[4] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010.

[5] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision*, 92(1):1–31, March 2011.

[6] Hynek Bakstein, Tomás Pajdla, and Daniel Vecerka. Rendering almost perspective views from a sparse set of omnidirectional images. In *BMVC*, pages 1–10, 2003.

[7] A. Baldacci, F. Ganovelli, M. Corsini, and R. Scopigno. Presentation of 3d scenes through video example. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.

[8] Andrea Baldacci, Daniele Bernabei, Massimiliano Corsini, Fabio Ganovelli, and Roberto Scopigno. 3d reconstruction for featureless scenes with curvature hints. *The Visual Computer*, pages 1–16, 2015.

[9] Andrea Baldacci, Fabio Ganovelli, Massimiliano Corsini, and Roberto Scopigno. Stereo-browsing from Calibrated Cameras. In Andrea Giachetti, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014.

[10] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[11] Sid Yingze Bao, Manmohan Chandraker, Yuanqing Lin, and Silvio Savarese. Dense Object Reconstruction with Semantic Priors. *CVPR*, pages 1264–1271, 2013.

[12] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[13] M. Bleyer, C. Rother, P. Kohli, D. Scharstein, and S. Sinha. Object stereo – joint stereo matching and object segmentation. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3081–3088, Washington, DC, USA, 2011. IEEE Computer Society.

[14] YY Boykov and MP Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. *ICCV 2001*, pages 105–112, 2001.

[15] Derek Bradley, Tamy Boubekeur, and Wolfgang Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. In *CVPR*. IEEE Computer Society, 2008.

[16] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial (2Nd Ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[17] Paolo Brivio, Luca Benedetti, Marco Tarini, Federico Ponchio, Paolo Cignoni, and Roberto Scopigno. Photocloud: Interactive remote exploration of joint 2d and 3d datasets. *Computer Graphics and Applications, IEEE*, 33(2):86–96, 2013.

[18] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proc. of the 8th European Conference on Computer Vision (ECCV2004)*, volume 3024, pages 25–36, 2004.

[19] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM, 2001.

[20] N.D.F. Campbell, G. Vogiatzis, C. Hernandez, and R. Cipolla. Automatic object segmentation from calibrated images. In *Visual Media Production Conference, 2011*, pages 126–137, Nov 2011.

[21] Neill D. Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Proceedings of the 10th European Conference on Computer Vision: Part I*, ECCV '08, pages 766–779, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] Gaurav Chaurasia, Sylvain Duchêne, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics*, 32(3):30:1–30:12, 2013.

[23] Gaurav Chaurasia, Olga Sorkine, and George Drettakis. Silhouette-aware warping for image-based rendering. *Computer Graphics Forum (Proceedings of the EUROGRAPHICS Symposium on Rendering)*, 30(4):1223—-1232, 2011.

[24] Shenchang Eric Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM, 1995.

[25] Mauro Cherubini, Rodrigo de Oliveira, and Nuria Oliver. Understanding near-duplicate videos: A user-centric approach. In *Proceedings of the 17th ACM International Conference on Multimedia (MM'09)*, pages 35–44, New York, NY, USA, 2009. ACM.

[26] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[27] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM, 1996.

[28] Marco Di Benedetto, Fabio Ganovelli, Marcos Balsa Rodriguez, Alberto Jaspe Villanueva, Roberto Scopigno, and Enrico Gobbetti. ExploreMaps: Efficient Construction and Ubiquitous Exploration of Panoramic View Graphs of Complex 3D Environments. *Computer Graphics Forum*, 2014.

[29] Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. A perceptual model for disparity. In *ACM Transactions on Graphics (TOG)*, volume 30, page 96. ACM, 2011.

[30] Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, Hans-Peter Seidel, and Wojciech Matusik. A luminance-contrast-aware disparity model and applications. *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia 2012, Singapour)*, 31(6), 2012.

[31] A. Djelouah, J.-S. Franco, E. Boyer, F. Le Clerc, and P. Perez. Multi-view object segmentation in space and time. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2640–2647, Dec 2013.

[32] Abdelaziz Djelouah, Jean-Sébastien Franco, Edmond Boyer, François Le Clerc, and Patrick Pérez. N-tuple color segmentation for multi-view silhouette extraction. In *ECCV (5)'12*, pages 818–831, 2012.

[33] Song-Pei Du, Belen Masia, Shi-Min Hu, and Diego Gutierrez. A metric of visual comfort for stereoscopic motion. *ACM Transactions on Graphics (TOG)*, 32(6):222, 2013.

[34] Frederic Dufaux, Béatrice Pesquet-Popescu, and Marco Cagnazzo. *Emerging Technologies for 3D Video: Creation, Coding, Transmission and Rendering*. John Wiley & Sons, 2013.

[35] Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson De Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating textures. In *Computer Graphics Forum*, volume 27, pages 409–418. Wiley Online Library, 2008.

[36] R. Fablet, P. Bouthemy, and P. Perez. Nonparametric motion characterization using causal probabilistic models for video indexing and retrieval. *Image Processing, IEEE Transactions on*, 11(4):393–407, Apr 2002.

[37] Christoph Fehn. A 3d-tv approach using depth-image-based rendering (dibr). In *Proc. of VIIP*, volume 3, 2003.

[38] Eduardo Fernández and Gonzalo Besuievsky. Inverse lighting design for interior buildings integrating natural and artificial sources. *Computers & Graphics*, 36(8):1096–1108, 2012.

[39] Eduardo Fernández and Gonzalo Besuievsky. Efficient inverse lighting: A statistical approach. *Automation in Construction*, 37:48–57, 2014.

[40] D. Freedman. Interactive Graph Cut Based Segmentation with Shape Priors. *CVPR'05*, 1:755–762, 2005.

[41] Y. Furukawa, B. Curless, S.M. Seitz, and R. Szeliski. Manhattan-world stereo. *CVPR2009*, pages 1422–1429, June 2009.

[42] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(8):1362–1376, aug. 2010.

[43] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Towards Internet-scale Multi-view Stereo. *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010.

[44] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148, 2015.

[45] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000.

[46] David Gallup, JM Frahm, and Marc Pollefeys. Piecewise planar and non-planar stereo for urban scene reconstruction. *CVPR'10*, pages 1418–1425, 2010.

[47] Anastasios Gkaravelis and Georgios Papaioannou. Inverse lighting design using a coverage optimization strategy. *The Visual Computer*, 32(6):771–780, 2016.

[48] Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, and Ronny Klowsky. Ambient point clouds for view interpolation. *ACM Transactions on Graphics (TOG)*, 29(4):95, 2010.

[49] Michael Goesele, Brian Curless, and Steven M. Seitz. Multi-view stereo revisited. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2402–2409, Washington, DC, USA, 2006. IEEE Computer Society.

[50] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 19(3):467–478, 2000.

[51] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.

[52] Hai Nam Ha and Patrick Olivier. *Lighting-by-Example with Wavelets*, pages 110–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[53] Arun Hampapur, Kiho Hyun, and Ruud M. Bolle. Comparison of sequence matching techniques for video copy detection. In *Proc. SPIE 4676, Storage and Retrieval for Media Databases 2002*, pages 194–201, 2001.

[54] Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 217–224, New York, NY, USA, 1996. ACM.

[55] Timothy C. Hoad and Justin Zobel. Fast video matching with signature alignment. In *Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR'03)*, pages 262–269, New York, NY, USA, 2003. ACM.

[56] Kenneth E. Hoff, III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 277–286, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[57] David M Hoffman, Ahna R Girshick, Kurt Akeley, and Martin S Banks. Vergence–accommodation conflicts hinder visual performance and cause visual fatigue. *Journal of vision*, 8(3):33–33, 2008.

[58] Maiya Hori, Masayuki Kanbara, and Naokazu Yokoya. Novel stereoscopic view generation by image-based rendering coordinated with depth information. In *Scandinavian Conference on Image analysis*, pages 193–202. Springer, 2007.

[59] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[60] Ian P Howard. *Seeing in depth, Vol. 1: Basic mechanisms.* University of Toronto Press, 2002.

[61] Peter A Howarth. Potential hazards of viewing 3-d stereoscopic television, cinema and computer games: a review. *Ophthalmic and Physiological Optics*, 31(2):111–122, 2011.

[62] Weiming Hu, Nianhua Xie, Li Li, Xianglin Zeng, and S. Maybank. A survey on visual content-based video indexing and retrieval. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(6):797–819, Nov 2011.

[63] Michal Jancosek and Tomas Pajdla. Segmentation based multi-view stereo. *Computer Vision Winter Workshop*, 2009.

[64] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In Andrew Zisserman David Forsyth, Philip Torr, editor, *European Conference on Computer Vision*, volume I of *LNCS*, pages 304–317. Springer, oct 2008.

[65] J.P.Guilford. *Psychometric Methods*. McGraw-Hill, 1954.

[66] Yong Ju Jung, Hosik Sohn, Seong-Il Lee, Hyun Wook Park, and Yong Man Ro. Predicting visual discomfort of stereoscopic images using human attention model. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(12):2077–2082, 2013.

[67] Kaveh Kardan and Henri Casanova. Virtual cinematography of group scenes using hierarchical lines of actions. In *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games (Sandbox'08)*, pages 171–178, New York, NY, USA, 2008. ACM.

[68] M Kass, A Witkin, and D Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, 1988.

[69] Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, pages 61–70, 2006.

[70] Sebastian Knorr, Matthias Kunter, and Thomas Sikora. Stereoscopic 3d from 2d video with super-resolution capability. *Signal Processing: Image Communication*, 23(9):665–676, 2008.

[71] Sebastian Knorr and Thomas Sikora. An image-based rendering (ibr) approach for realistic stereo view synthesis of tv broadcast based on structure from motion. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 6, pages VI–572. IEEE, 2007.

[72] K. Kolev, M. Klodt, T. Brox, and D. Cremers. Propagated photoconsistency and convexity in variational multiview 3d reconstruction. In *Workshop on Photometric Analysis for Computer Vision*, Rio de Janeiro, Brazil, oct 2007.

[73] K. Kolev, T. Pock, and D. Cremers. Anisotropic minimal surfaces integrating photoconsistency and normal information for multiview stereo. In *ECCV'10*, Heraklion, Greece, September 2010.

[74] Kalin Kolev, Thomas Brox, and Daniel Cremers. Robust variational segmentation of 3D objects from multiple views. *Pattern Recognition*, pages 688–697, 2006.

[75] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):147–59, February 2004.

[76] Janusz Konrad. View reconstruction for 3-d video entertainment: issues, algorithms and applications. In *Image Processing And Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465)*, volume 1, pages 8–12. IET, 1999.

[77] Marc Lambooij, Marten Fortuin, Ingrid Heynderickx, and Wijnand IJsselsteijn. Visual discomfort and visual fatigue of stereoscopic displays: a review. *Journal of Imaging Science and Technology*, 53(3):30201–1, 2009.

[78] Manuel Lang, Alexander Hornung, Oliver Wang, Steven Poulakos, Aljoscha Smolic, and Markus Gross. Nonlinear disparity mapping for stereoscopic 3d. *ACM Transactions on Graphics (TOG)*, 29(4):75, 2010.

[79] C H Lee, Amitabh Varshney, and D W Jacobs. Mesh saliency. In *ACM SIGGRAPH 2005 Papers*, volume 24, pages 659–666. ACM, July 2005.

[80] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. *ACM Trans. Graph.*, 24(3):659–666, July 2005.

[81] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM, 1996.

[82] J. Lezama, K. Alahari, J. Sivic, and I. Laptev. Track to the future: Spatio-temporal video segmentation with long-range motion cues. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR2011)*, pages 3369–3376, June 2011.

[83] Haibin Ling and Kazunori Okada. Emd-l 1: an efficient and robust algorithm for comparing histogram-based descriptors. In *Computer Vision–ECCV 2006*, pages 330–343. Springer, 2006.

[84] Jiajun Liu, Zi Huang, Hongyun Cai, Heng Tao Shen, Chong Wah Ngo, and Wei Wang. Near-duplicate video retrieval: Current research and future trends. *ACM Comput. Surv.*, 45(4):44:1–44:23, August 2013.

[85] Yu-Fei Ma and Hong-Jiang Zhang. Motion texture: a new motion based video representation. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 548–551, 2002.

[86] EN Mortensen and WA Barrett. Intelligent scissors for image composition. *Proceedings of the 22nd annual conference ...*, 84602(801), 1995.

[87] Marius Muja and David G. Lowe. Fast matching of binary features. In *Proceedings of the Ninth Conference on Computer and Robot Vision (CRV'12)*, pages 404–410, Washington, DC, USA, 2012. IEEE Computer Society.

[88] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[89] T. Mörwald, J. Balzer, and M. Vincze. Direct optimization of t-splines based on multiview stereo. In *2014 2nd International Conference on 3D Vision*, volume 1, pages 20–27, Dec 2014.

[90] Liangliang Nan, Andrei Sharf, and Baoquan Chen. 2D-3D Lifting for Shape Reconstruction. *Computer Graphics Forum*, 33(7):249–258, 2014.

[91] HoangMinh Nguyen, Burkhard Wünsche, Patrice Delmas, Christof Lutteroth, and Eugene Zhang. A robust hybrid image-based modeling system. *The Visual Computer*, pages 1–16, 2015.

[92] Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'09)*, pages 55–65, New York, NY, USA, 2009. ACM.

[93] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Comp. Graph. Forum*, 28(2):493–501, 2009.

[94] Fabio Pellacini. envylight: an interface for editing natural illumination. In *ACM Transactions on Graphics (TOG)*, volume 29, page 34. ACM, 2010.

[95] Fabio Pellacini, Frank Battaglia, R. Keith Morley, and Adam Finkelstein. Lighting with paint. *ACM Trans. Graph.*, 26(2), June 2007.

[96] Dimitri Plemenos, Jérôme Grasset, Benoît Jaubert, and Karim Tamine. Intelligent visibility-based 3D scene processing techniques for computer games. In *Proc. of GraphiCon 2005 - International Conference on Computer Graphics and Vision*, 2005.

[97] Christian Richardt, Lech Świrski, Ian P Davies, and Neil A Dodgson. Predicting stereoscopic viewing comfort using a coherence-based computational model. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, pages 97–104. ACM, 2011.

[98] Carsten Rother and V Kolmogorov. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (*, 2004.

[99] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:2000, 2000.

[100] Waqar Saleem, Wenhao Song, Alexander Belyaev, and Hans-Peter Seidel. On computing best fly. In *Proceedings of the 23rd Spring Conference on Computer Graphics*, SCCG '07, pages 115–121, New York, NY, USA, 2007. ACM.

[101] Brian Salomon, Maxim Garber, Ming C. Lin, and Dinesh Manocha. Interactive navigation in complex environments using path planning. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (I3D'03)*, pages 41–50, New York, NY, USA, 2003. ACM.

[102] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting and material. In *Computer Graphics Forum*. Wiley Online Library, 2015.

[103] Adrian Secord, Jingwan Lu, Adam Finkelstein, Manish Singh, and Andrew Nealen. Perceptual models of viewpoint preference. *ACM Transactions on Graphics (TOG)*, 30(5):109, 2011.

[104] Adrian Secord, Jingwan Lu, Adam Finkelstein, Manish Singh, and Andrew Nealen. Perceptual models of viewpoint preference. *ACM Transactions on Graphics*, 30(5):1–12, October 2011.

[105] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR2006)*, volume 1, pages 519–528, june 2006.

[106] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073, 1997.

[107] A. SenGupta, D. M. Thounaojam, K. M. Singh, and S. Roy. Video shot boundary detection: A review. In *Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on*, pages 1–6, March 2015.

[108] SN Sinha, Drew Steedly, and Richard Szeliski. Piecewise planar stereo for image-based rendering. *ICCV*, pages 1881–1888, September 2009.

[109] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. *ACM transactions on graphics (TOG)*, 25(3):835–846, 2006.

[110] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006.

[111] Dmitry Sokolov, Dimitri Plemenos, and Karim Tamine. Methods and data structures for virtual world exploration. *The Visual Computer*, 22(7):506–516, 2006.

[112] Dmitry Sokolov, Dimitri Plemenos, and Karim Tamine. Viewpoint quality and global scene exploration strategies. *GRAPP*, 2006:184–191, 2006.

[113] M. Sormann, C. Zach, and K. Karner. Graph cut based multiple view segmentation for 3d reconstruction. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 1085–1092, June 2006.

[114] Mario Sormann, Christopher Zach, and Konrad Karner. Graph Cut Based Multiple View Segmentation for 3D Reconstruction. *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 1085–1092, June 2006.

[115] William Thompson, Roland Fleming, Sarah Creem-Regehr, and Jeanine Kelly Stefanucci. *Visual perception from a computer graphics perspective*. CRC Press, 2011.

[116] PP Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint Selection using Viewpoint Entropy. In *Proceedings of the Vision Modeling and Visualization Conference (VMV2001)*, 2001.

[117] Sara Vicente. Graph cut based image segmentation with connectivity priors. *CVPR'08*, pages 1–8, 2008.

[118] G. Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, 1990.

[119] Kouki Watanabe and Alexander G Belyaev. Detection of salient curvature features on polygonal surfaces. In *Computer Graphics Forum*, volume 20, pages 385–392. Wiley Online Library, 2001.

[120] D White K. . Cline and P Egbert. Poisson disk point sets by hierarchical dart throwing. *Symposium on Interactive Ray Tracing*, pages 129–132, 2007.

[121] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.

[122] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M. Seitz. Schematic surface reconstruction. *CVPR2012*, pages 1498–1505, 2012.

[123] Changchang Wu et al. Visualsfm: A visual structure from motion system.

[124] K. Yamaguchi, H. Takemura, K. Yamazawa, and N. Yokoya. Real-time generation and presentation of view-dependent binocular stereo images using a sequence of omnidirectional images. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 589–593 vol.4, 2000.

[125] Mingjin Yan. *Methods of determining the number of clusters in a data set and a new clustering criterion*. PhD thesis, Virginia Tech, 2005.

[126] Anthony Yezzi and S Soatto. Stereoscopic segmentation. *IJCV*, 53(1):31–43, 2003.

[127] Alexander Zaslavski and M Powell. The NEWUOA software for unconstrained optimization without derivatives. In Panos Pardalos, G Pillo, and M Roma, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, chapter 16, pages 255–297. Springer US, Boston, 2006.

[128] Wenlan Zhang, Ting Luo, Gangyi Jiang, Qiuping Jiang, Hongwei Ying, and Jing Lu. Using saliency-weighted disparity statistics for objective visual comfort assessment of stereoscopic images. *3D Research*, 7(2):1–11, 2016.

[129] Ke Colin Zheng, Alex Colburn, Aseem Agarwala, Maneesh Agrawala, David Salesin, Brian Curless, and Michael F. Cohen. Parallax photography: Creating 3d cinematic effects from stills. In *Proceedings of Graphics Interface 2009 (GI'09)*, pages 111–118, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[130] Zihan Zhou, Hailin Jin, and Yi Ma. Plane-based content preserving warps for video stabilization. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2299–2306. IEEE, 2013.

[131] Chen Zhu and WeeKheng Leow. Textured mesh surface reconstruction of large buildings with multi-view stereo. *The Visual Computer*, 29(6-8):609–615, 2013.